Tasks for Doc or Support Agent

□ Upload & Vector DB Integration

Story 1: Configurable Upload to Vector DB with Future-Proofing → Assignee: → AGAI-1e2165274-dd7f-3533-9e5b-e56e11daee4bSystem Jira

- Develop an API endpoint to ingest content along with vector DB configuration details.
- Support inputs from:
  - Manual file upload(s)
  - Confluence content via Confluence Fetcher
  - SharePoint content via SharePoint Fetcher
- Implement chunking logic with configurable chunk size and overlap percentage.
- Maintain a metadata table to track file uploads, source, chunk count, and storage mapping.
- Abstract vector DB interface so switching between OpenAI, Oracle23AI, Pinecone, etc., requires only config change.
- Include content preprocessing (format normalization, encoding checks) before vectorization.
- **Note**: One thought we can check if we can use https://github.com/zcaceres/markdownify-mcp and convert all input types to markdown and the API can take markdown as input.

Story 2: Manual Upload Support for Multiple Document Types → Assignee:

- Develop an interface/API to allow uploading one or more documents of supported formats (PDF, DOCX, TXT, HTML, Markdown).
- Integrate with the **MarkItDown MCP server** to convert uploaded files into a consistent Markdown format. Check : https://github.com/zcaceres/markdownify-mcp
- Use the converted Markdown files as input to the generic upload API for Vector DB ingestion, ensuring seamless handling of multiple document types.
- Route the uploaded content through the **generic upload API** developed in **Story 1**.
- Implement automatic tagging to associate each document with the correct product/category based on provided metadata.

Story: Integrate MarkItDown MCP for Universal Markdown Conversion

Linked Story: AGAI-1 (Configurable Upload to Vector DB with Future-Proofing)

Description:

Integrate and configure the **MarkItDown MCP server** to convert various input content formats into clean Markdown. This Markdown will serve as a unified input format for the Vector DB ingestion API (developed in AGAI-1). The goal is to ensure consistent, scalable content preprocessing across formats and systems.

Story 3: Configuration-Driven Upload for SharePoint & Confluence

- Use `config.json` files or admin UI to manage SharePoint/Confluence sync settings.
- Include:
  - Site/Page IDs
  - Fetch frequency
  - File type rules
- Ensure all content is routed to the upload API from Story 1 for chunking, processing, and vectorization.

Story 4: UI for On-Demand Upload with Confluence Page Input

- Provide a dedicated UI to input:
  - Confluence space or page ID
  - Option to include/exclude child pages
- Trigger download, extraction, and upload using APIs from previous stories.

Story 19: MetricStream University Videos → Transcript Generation → Knowledge Base Integration (Vector DB)

- Provide an option to add MetricStream University Learning Videos as input.
- Integrate or develop functionality to automatically generate **transcripts** from the video content, capturing metadata (e.g., title, duration, speaker, tags).
- Convert generated transcripts into a structured format (Markdown or plain text) using **MarkItDown MCP** or equivalent tooling.
- Upload the processed transcripts into the **VectorDB** to enhance KnowledgeBase search and chatbot query responses.
- When users query related topics:
  - Retrieve relevant transcript information for better, more accurate answers.
  - If applicable, provide a **playable video link** along with the response for deeper understanding.
- This approach will improve user experience by reducing the need to manually search videos and ensuring direct access to precise, contextually linked video materials.

□ Chatbot Enhancements

Story 5: Cleanup of Chatbot Response Format Issues → Assignee: → AGAI-2e2165274-dd7f-3533-9e5b-e56e11daee4bSystem Jira

- Fix:
  - Triple quote (`'''`) artifacts
  - HTML tags rendering in plain copy/paste scenarios
  - PDF export formatting (table header overlaps, title not wrapping)
- Add filters to clean/format markdown responses before rendering/export.

- Consider other issues in Chatbot.

Story 6: UX Enhancements for Chatbot & Report Screen

- Improve chatbot and report interfaces with:
    - Tooltips
    - Consistent font rendering
    - Accessibility improvements
    - Scroll-to-top, loading states, retry buttons

Story 7: Hallucination Detection and Correction in Chatbot Responses → Assignee:

- Detect low confidence/hallucinated responses using heuristics or model confidence.
- Allow:
    - Retry/rephrase
    - User thumbs up/down
    - Text feedback input
- Feed responses to an internal review system to improve prompts or filter responses.

Story 8: Cross-Agent Communication Between Product Chatbots → Assignee:

- Enable inter-agent messaging: Product-specific chatbots should talk to each other.
- Consolidate responses across multiple product domains.
- Clearly show source chatbot in the unified answer.

□ Report Generation

Story 9: Refine Prompts & Instructions for Change History & Feature Evolution Reports → Assignee: → AGAI-3e2165274-dd7f-3533-9e5b-e56e11daee4bSystem Jira

- Audit existing vs old reports for completeness and accuracy.
- Refine prompts and context-setting instructions to produce more relevant output.
- Reference SharePoint documentation and prior assistant responses.

Story 10: Optimize Formatting Flow for Reports Without ChatGPT Call → Assignee:

- Assess feasibility of replacing the second ChatGPT call (for formatting).
- Implement server-side markdown-to-PDF formatting using open-source libraries (e.g., `markdown-pdf`, `WeasyPrint`, or HTML+CSS engine).

Story 11: Implement RLHF for Report Generation → Assignee:

- Enable end-users to rate reports and add feedback.
- Track rating history.
- Use feedback to adjust prompt phrasing, example patterns, and formatting logic.

□ System Integration & Automation

Story 12: Enhance Confluence Auto Download (Images, Tables, Links), MCP Integration for Confluence - Delta Sync → Assignee: → AGAI-5e2165274-dd7f-3533-9e5b-e56e11daee4bSystem Jira

- Enhance current parser to extract:
  - Embedded images
  - Tables (including nested)
  - Cross-linked pages and anchors
- Store enriched output in vector format preserving contextual structure.
- Compare version metadata (modified timestamp, version number).
- Download and vectorize only the changed Confluence pages.
- Update tracking metadata post-processing.

Story 13: SharePoint Auto Download & Delta Download with MCP Integration → Assignee: → AGAI-6e2165274-dd7f-3533-9e5b-e56e11daee4bSystem Jira

- Implement MCP-based automation for SharePoint auto download based on config.
- Fetch pages/files, extract rich content (images, tables, links).
- Feed to vector pipeline.
- Identify modified documents based on timestamps or hash comparison.
- Only pull deltas, skipping unchanged files.
- Integrate with version tracking table.

Story 14: JIRA Integration via MCP Server → Assignee:

- Fetch JIRA issues, comments, updates using Atlassian API.
- Implement delta sync and indexing logic.
- Track issues per product/module for vector enhancement.

□ Other Enhancements

Story 15: Configure OpenAI Assistants in Doc Agent for other Products

- Allow registration of product-specific assistants.
- Support separate prompts, embedding types, and document selectors.
- Enable Assistant switching logic based on user input domain.

Story 16: Log Analyzer - Pattern & Issue Detection → Assignee:

- Develop a UI/API to upload logs (App/DB/Server).
- Parse log levels, timestamps, error codes.
- Use basic NLP or regex to group similar errors and identify frequent failures.

□□ Base Framework Changes

Story 17: Folder Structure Enhancement → Assignee: → AGAI-4e2165274-dd7f-3533-9e5b-e56e11daee4bSystem Jira

- Split server-side and client-side folders clearly.
- Introduce folders for:
    - Each type of Agent (Upload Agent, Chat Agent, Report Agent)
    - Common utilities
- Ensure project structure supports easy scaling and modular code.

Story 18: Migrate to FastAPI Framework → Assignee:

- Evaluate feasibility of replacing Flask with Fast API.
- Refactor existing APIs to FastAPI syntax.
- Benefits:
    - Better performance (async support)
    - Built-in docs with OpenAPI
    - Type hinting for better dev experience