Agentic Framework Implementation

This document outlines the implementation of the agentic framework in the PSScript platform, which uses the OpenAI Assistants API to provide a more powerful and persistent AI experience.

Overview

The PSScript platform has been enhanced with an agentic framework that leverages the OpenAI Assistants API. This implementation follows the principles outlined in the OpenAI Assistants API documentation.

Key features of this implementation:

1. **Persistent Conversations**: The system maintains conversation state across user interactions using session IDs.
2. **Specialized Agents**: The framework supports different agent types, with the default being the OpenAI Assistants API.
3. **Enhanced Context**: The agents can maintain context over long conversations, improving the quality of responses.

Architecture

The agentic framework is implemented across several components:

Backend (Node.js/Express)

- **ChatController**: Handles incoming chat requests and forwards them to the AI service.
- Added support for `agent_type` and `session_id` parameters to specify which agent to use and maintain conversation state.

AI Service (Python)

- **OpenAI Assistant Agent**: Implements the OpenAI Assistants API integration.
- **Agent Coordinator**: Manages different agent types and routes requests to the appropriate agent.
- **Main API**: Exposes endpoints for chat, analysis, and other AI functions.

Frontend (React)

- **useChat Hook**: Custom hook that manages chat state and communication with the backend.
- **SimpleChatWithAI**: Component that provides the chat interface.
- Added support for maintaining session state across conversations.

Implementation Details

OpenAI Assistant Agent

The `OpenAIAssistantAgent` class in `src/ai/agents/openai_assistant_agent.py` implements the OpenAI Assistants API integration. Key features:

- **Thread Management**: Creates and manages threads for persistent conversations.
- **Assistant Configuration**: Configures the assistant with appropriate instructions and tools.
- **Message Processing**: Handles sending messages to the assistant and processing responses.

Chat Controller

The `ChatController` in `src/backend/src/controllers/ChatController.ts` has been updated to:

- Accept `agent_type` and `session_id` parameters.
- Forward these parameters to the AI service.
- Return the `session_id` in the response for persistent conversations.

Frontend Integration

The frontend has been updated to:

- Store and manage the `sessionId` in the `useChat` hook.
- Pass the `sessionId` to the backend when sending messages.
- Update the `sessionId` when receiving a response with a new session ID.

Usage

To use the agentic framework:

1. The frontend sends a message to the backend with an optional `agent_type` parameter (defaults to "assistant").
2. The backend forwards the request to the AI service with the specified agent type.
3. The AI service routes the request to the appropriate agent.
4. The agent processes the message and returns a response.
5. The response includes a `session_id` that can be used for future messages to maintain conversation state.

Configuration

The agentic framework can be configured through environment variables:

- `OPENAI_API_KEY`: API key for OpenAI services.
- `OPENAI_ASSISTANT_ID`: ID of a pre-created assistant to use (optional).

Future Enhancements

Planned enhancements to the agentic framework include:

1. **Tool Integration**: Adding support for custom tools that the assistant can use.
2. **Multi-Agent Collaboration**: Enabling multiple agents to collaborate on complex tasks.
3. **Enhanced Memory**: Implementing more sophisticated memory mechanisms for longer context.
4. **Task Planning**: Adding support for breaking down complex tasks into smaller steps.
5. **State Visualization**: Providing visualizations of the agent's internal state.

References

- OpenAI Assistants API Documentation
- OpenAI Function Calling
- OpenAI Thread Management

Generated 2026-01-16 21:23 UTC