Software Flow Diagram:

Inputs:
Voice Commands from User (listen_to_activation_word working to hear all the inputs)

Outputs:
Audio Output from the Speaker

(None of the classes are fully developed, though there is existing code for VoiceCapture and VoiceProcessor)

Classes & Functions:
1. **Class: VoiceCapture**
- Purpose: To interface with the microphone and capture user's voice.
  - `start_capture()`: Initiates voice recording.
  - `stop_capture()`: Stops voice recording.
  - `get_audio_data()`: Retrieves the latest audio data captured.

2. **Class: VoiceProcessor**
- Purpose: To convert voice to text and vice versa.
  - `voice_to_text(audio_data)`: Converts voice data to text using a VTT Service. Returns the transcribed text.
  - `text_to_voice(text_data)`: Converts text to voice using a TTS Service. Returns audio data.

3. **Class: GPT_API_Interface**
- Purpose: To communicate with the GPT API.
  - `send_request(text_data)`: Sends the text data to GPT API for processing. Returns GPT's response.
  - `get_response()`: Retrieves the latest response from the GPT API.

4. **Class: AudioOutput**
- Purpose: To handle audio output operations.
  - `play_audio(audio_data)`: Plays the provided audio data through the speaker.
  - `stop_audio()`: Stops any ongoing audio output.

5. **Class: WiFiManager**
- Purpose: To manage the WiFi connection.
  - `connect_to_wifi()`: Establishes a connection to the WiFi network.
  - `disconnect_from_wifi()`: Disconnects from the WiFi network.
  - `check_connection_status()`: Checks and returns the current connection status.

6. **Class: MainController**
- Purpose: To orchestrate the overall flow and operations.

- `init_system()`: Initializes system components (e.g., WiFi connection, microphone).
- `capture_voice_input()`: Invokes voice capture and sends it for processing.
- `process_voice()`: Uses VoiceProcessor for VTT and invokes GPT API Interface.
- `get_gpt_response()`: Fetches GPT's response and sends it to TTS.
- `output_response()`: Sends TTS response to the speaker for audio output.

Possible supplemental functions:

****User Interaction**:**
- `listen_for_activation_word()`: Listens continuously and activates the system when a specific word or phrase (e.g., "Hey GPT") is detected.
- `provide_feedback(feedback_type)`: Provides audible feedback for actions like start listening, error encountered, processing, etc.

****Configuration & Settings**:**
- `load_configuration()`: Loads user-specific settings (like preferred language, volume levels, etc.) from a stored file or database.
- `update_configuration(new_settings)`: Allows updating and saving of user-specific settings.
- `set_volume(level)`: Adjusts the speaker volume.

****Battery & Power Management****
- `check_battery_status()`: Checks and returns current battery level.
- `enter_power_saving_mode()`: Reduces power consumption when not in active use.
- `wake_up_from_power_saving()`: Wakes the device up from power-saving mode.

****Network & API Handling**:**
- `handle_api_rate_limits()`: Slows down or stops requests when rate limits are approached or exceeded.
- `validate_network_latency()`: Checks for network speed and decides if operations can proceed smoothly.

****Data Management**:**
- `store_local_cache(data)`: Stores recent queries/responses locally for faster retrieval, especially useful if the same questions are asked frequently.
- `clear_local_cache()`: Deletes the local cache, useful for privacy or troubleshooting.
- `anonymize_data(data)`: Removes or alters personal information from data before sending to GPT or storing.

Sequential Flow of Information and Interaction of Functions:

1. **Initialization:**
- **MainController** -> **WiFiManager**: `connect_to_wifi()`
- **MainController** -> **AudioOutput**: `set_volume(level)`
- **MainController** -> **Configuration & Settings**: `load_configuration()`

2. **Wait for Activation:**
  - **MainController** -> **User Interaction**: `listen_for_activation_word()`

3. **Once Activated:**
  - **MainController** -> **VoiceCapture**: `start_capture()`
  - **MainController** -> **VoiceCapture**: `stop_capture()`
  - **MainController** -> **VoiceCapture**: `get_audio_data()`

4. **Voice to Text Conversion:**
  - **MainController** -> **VoiceProcessor**: `voice_to_text(audio_data)`

5. **Send Query to GPT API:**
  - **MainController** -> **GPT_API_Interface**: `send_request(text_data)`

6. **Retrieve Response from GPT API:**
  - **MainController** -> **GPT_API_Interface**: `get_response()`

7. **Convert Response to Voice:**
  - **MainController** -> **VoiceProcessor**: `text_to_voice(text_data)`

8. **Play Response:**
  - **MainController** -> **AudioOutput**: `play_audio(audio_data)`

9. **Error Handling (occurs at any step if an error is detected):**
  - **MainController** -> **Error Management**: `handle_errors(error_code)`

10. **Log Events (Optional, can be invoked at multiple stages for logging purposes):**
  - **MainController** -> **Logging & Monitoring**: `log_event(event_type, details)`

11. **Check for Updates (Can be periodically invoked):**
  - **MainController** -> **Updates & Maintenance**: `check_for_updates()`

12. **Handle Connection Issues:**
  - **MainController** -> **WiFiManager**: `check_connection_status()`
  - **MainController** -> **WiFiManager**: `retry_last_operation()`

13. **Return to Activation Listening:**
  - Loop back to Step 2.