

DOKUMENTATION: PROTA-BOT

Stand: 13.12.2024 – Änderungen vorbehalten.

Programm: Christoph Martsch-Grunau (@elektropastor)

Inhaltliche Autorenschaft: Lars Henriks, Moritz Haase

INHALT

| | |
|--|-----------|
| INHALT | 2 |
| ÜBER DEN PROTA-BOT | 3 |
| HAUPTKOMPONENTEN | 4 |
| 1. multi_bot.py | 4 |
| 2. multi_rag.py | 7 |
| 3. multi_logger.py | 9 |
| 4. multi_settings.py | 11 |
| 5. multi_config.py | 14 |
| KERNFUNKTIONEN | 16 |
| Initialisierung | 16 |
| Logging | 16 |
| Interaktionen | 16 |
| RAG-System | 16 |
| CHARAKTEREINSTELLUNGEN | 17 |
| Thomas Eberle: Der überhebliche Elektronikunternehmer | 17 |
| Martin: Die verzweifelte Seele in einer Kaffeemaschine | 18 |
| Erweiterbarkeit | 19 |
| INSTALLATION UND NUTZUNG | 20 |
| Python-Pakete | 20 |
| Dateien | 20 |
| Verzeichnisse | 21 |
| Drittanbieter-APIs | 21 |
| Systemvoraussetzungen | 21 |
| Installation | 21 |
| Installations-Schritte | 22 |
| Fehlerbehebung | 22 |
| ERWEITERUNG DES PROTA-BOTS | 23 |
| Hinzufügen neuer Charaktere | 23 |
| Integration weiterer Funktionen | 23 |
| Fazit | 24 |

ÜBER DEN PROTA-BOT

Der Prota-Bot ist ein vielseitiger Discord-Bot, der entwickelt wurde, um Server mit unterhaltsamen, interaktiven und informativen Inhalten zu bereichern. Ein zentrales Feature ist seine Fähigkeit, verschiedene Charaktere mit einzigartigen Persönlichkeiten und Funktionen zu simulieren. Dies macht ihn ideal für Rollenspiele, Community-Interaktionen oder kreative Anwendungen.

Mithilfe von OpenAI-Technologien generiert der Bot intelligente und kontextbezogene Antworten, die durch das Retrieval-Augmented-Generation (RAG)-System weiter optimiert werden. Dieses System integriert relevante externe Informationen, wodurch die Antworten präziser und tiefgründiger werden. Der Prota-Bot verbindet somit technologische Innovation mit ansprechender Unterhaltung und bietet Benutzern ein personalisiertes und interaktives Erlebnis. Seine Anpassbarkeit ist ein großer Vorteil, da Administratoren ihn flexibel auf die Bedürfnisse ihrer Community zuschneiden können.

Auf dem Discord-Server „Antiverse“ speist sich der Bot aus speziell aufbereiteten Drehbüchern des Hörspiels „Korridore“ (SWR, Autoren: Lars Henriks und Moritz Haase). Bisher sind die Protagonisten „Thomas Eberle“ und die „Kaffeemaschine Martin“ implementiert. Für ihren jeweils einzigartigen Charakter wurden spezifische Systemmessages konzipiert und von den Drehbuchautoren ausgestaltet.

HAUPTKOMPONENTEN

Dieser Abschnitt bietet eine strukturierte Übersicht über die Kernmodule und deren jeweilige Funktionalitäten. Ziel ist es, die Architektur des Prota-Bots verständlich darzustellen und die zentrale Rolle jeder Komponente innerhalb des Systems hervorzuheben.

1. multi_bot.py

Die Datei `multi_bot.py` ist das Hauptskript, das den Prota-Bot steuert. Sie bietet eine umfassende Verwaltung des Bots, von der Initialisierung bis hin zur Interaktion mit Nutzern auf Discord-Servern. Das Skript umfasst die grundlegenden Logiken und Funktionen des Bots, darunter:

- Verwaltung von Charakteren.
- Interaktionshandling.
- Logging.
- Integration mit der OpenAI-API.

Hauptfunktion: BotManager

Die Hauptklasse `BotManager` ist für die Steuerung des Prota-Bots verantwortlich. Sie wird mit einem spezifischen Charakter gestartet, der über die Kommandozeile angegeben wird.

Initialisierung

- Beim Start des Bots wird überprüft, ob der gewählte Charakter in den Konfigurationen hinterlegt ist.
- Die Klasse übernimmt die Einstellungen aus den Dateien `multi_settings.py` und `multi_config.py`.
- Logging wird über die `LogManager`-Klasse konfiguriert, wobei für jeden Charakter spezifische Log-Dateien erstellt werden.

Wichtige Methoden

- `setup_events()`: Definiert die Events für den Bot, z. B. `on_ready` und `on_message`. Damit wird sichergestellt, dass der Bot korrekt auf Ereignisse reagiert.
- `setup_commands()`: Stellt benutzerdefinierte Befehle bereit, die von Administratoren oder Nutzern ausgeführt werden können.
- `run()`: Startet den Bot mit dem entsprechenden Token.

Nutzen für Benutzer

- Benutzer können einfach mit dem Bot interagieren, indem sie Nachrichten schreiben oder Befehle nutzen.
- Administratoren können die Logs überprüfen, um die Aktivitäten des Bots zu verfolgen.

Interaktion: InteractionHandler

Die Klasse `InteractionHandler` verarbeitet alle Benutzerinteraktionen und ist das Herzstück der Kommunikation mit der OpenAI-API. Sie sorgt für:

- Die Verarbeitung von Benutzeranfragen.
- Die Verwaltung des Nachrichtenverlaufs.
- Die Integration von Antworten der KI in die Benutzerinteraktionen.

Nachrichtenverlauf

Der Nachrichtenverlauf wird für jeden Benutzer in einem Kanal separat gespeichert. Dies ermöglicht eine kontextbezogene Antwort.

Wichtige Methoden

- `update_channel_history()`: Speichert und aktualisiert die letzten Nachrichten im Verlauf.
- `handle_interaction()`: Die Hauptmethode, um Benutzeranfragen zu verarbeiten. Sie generiert Antworten mithilfe von OpenAI und sendet diese an den Kanal zurück.
- `summarize_messages()`: Erstellt Zusammenfassungen vergangener Nachrichten, um der KI den Kontext für Antworten zu geben.

Nutzen für Benutzer

- Benutzer erhalten intelligente und kontextbezogene Antworten.
- Der Bot kann auf den Verlauf eingehen, um präzisere Reaktionen zu liefern.

Charakterverwaltung: CharacterManager

Die Klasse `CharacterManager` sorgt dafür, dass der Bot je nach gewähltem Charakter korrekt konfiguriert ist.

Aufgaben

- Laden der Charaktereinstellungen aus `CHARACTER_SETTINGS` in `multi_settings.py`.
- Bereitstellen einer spezifischen Systemnachricht, die an die OpenAI-API übergeben wird.

Nutzen für Benutzer

- Der Bot verhält sich entsprechend der gewählten Charakterpersönlichkeit.
- Die Antworten spiegeln die definierten Eigenschaften des Charakters wider.

Logging: LogManager

Die Klasse `LogManager` konfiguriert das Logging für den Prota-Bot. Sie erstellt charakter-spezifische Log-Verzeichnisse und Dateien.

Aufgaben

- Sicherstellen, dass alle Ereignisse protokolliert werden.
- Verwaltung von rotierenden Log-Dateien, um die Speichernutzung zu optimieren.

Nutzen für Benutzer

Administratoren können Logs nutzen, um Probleme zu analysieren oder die Interaktionen nachzuvollziehen.

Technische Highlights

JSON-zu-Markdown-Konvertierung

Die Funktion `json_to_markdown` wandelt JSON-Daten in ein Markdown-Format um. Dies ist nützlich für die Übersichtlichkeit von Logs oder KI-Antworten.

OpenAI-Integration

- Der Bot nutzt die OpenAI-API, um Antworten auf Benutzeranfragen zu generieren.
- Die Antworten werden auf Basis von Kontext, Verlauf und definierten Charaktereigenschaften erstellt.

Fehlerbehandlung

Das Skript enthält robuste Mechanismen zur Fehlererkennung und -behandlung, z. B. bei API-Zeitüberschreitungen oder ungültigen Eingaben.

Nutzung des Prota-Bots

Interaktion:

- Schreibe eine Nachricht im entsprechenden Discord-Kanal.
- Erwähne den Charakter mit @.

Fehlerbehebung:

- Prüfe die Logs im Verzeichnis `logs/<Charaktername>/`.
- Stelle sicher, dass die Tokens und API-Schlüssel korrekt konfiguriert sind.

2. multi_rag.py

Die Datei `multi_rag.py` implementiert das Retrieval-Augmented-Generation (RAG)-System des Prota-Bots. Sie dient dazu, relevante Informationen aus gespeicherten Textdaten zu extrahieren und diese kontextbezogen in die Antworten der KI zu integrieren. Dadurch können Antworten sowohl auf Grundlage des Nachrichtenverlaufs als auch externer Daten generiert werden.

Wesentliche Funktionen der Datei:

- Erstellung und Verwaltung von FAISS-Indizes.
- Verarbeitung und Analyse von Benutzerfragen.
- Integration von Hintergrundinformationen in die KI-Antworten.

Hauptfunktion: FAISSManager

Die Klasse `FAISSManager` ist verantwortlich für das Laden, Erstellen und Verwalten von FAISS-Indizes. Diese Indizes werden verwendet, um relevante Informationen effizient abzurufen.

Initialisierung

- Der Manager erstellt einen charakter-spezifischen Datenpfad, in dem die FAISS-Indizes gespeichert werden.
- Die Klasse verwendet OpenAI-Embeddings, um Textdaten in Vektoren umzuwandeln.

Wichtige Methoden

- `load_txt_files()`: Lädt Textdateien aus einem definierten Datenverzeichnis.
- `create_index()`: Erstellt einen neuen FAISS-Index aus den geladenen Textdaten.
- `load_or_create_index()`: Prüft, ob ein FAISS-Index existiert. Falls nicht, wird ein neuer Index erstellt.

Nutzen für Benutzer

- Ermöglicht es dem Bot, auf umfangreiche Datensammlungen zuzugreifen.
- Sicherstellt, dass der Bot konsistente und fundierte Antworten liefern kann.

Datenabruf: RAGSystem

Die Klasse `RAGSystem` stellt die Verbindung zwischen dem FAISS-Index und der Beantwortung von Benutzerfragen her. Sie ruft relevante Informationen aus dem Index ab und integriert diese in die KI-Antworten.

Wichtige Methoden

- `retrieve_context()`: Ruft kontextbezogene Daten aus dem FAISS-Index ab, basierend auf der Benutzeranfrage.

Nutzen für Benutzer

- Liefert kontextbezogene Informationen, die Antworten präziser und informativer machen.
- Macht den Prota-Bot zu einem leistungsstarken Werkzeug für komplexe Anfragen.

Frageverarbeitung: QuestionProcessor

Die Klasse `QuestionProcessor` verbindet das RAG-System mit der OpenAI-API, um Benutzerfragen zu beantworten. Sie kombiniert Informationen aus dem Nachrichtenverlauf, dem FAISS-Index und der aktuellen Anfrage.

Wichtige Methoden

- `answer_question()`: Bearbeitet Benutzeranfragen, ruft relevante Informationen ab und generiert Antworten.

Nutzen für Benutzer

- Nutzer profitieren von Antworten, die sowohl auf vorhandene Daten als auch auf kontextuelle Informationen eingehen.
- Die Antworten sind fundierter und nachvollziehbar.

Technische Highlights

Asynchrone Verarbeitung

- Die Datei verwendet asynchrone Operationen, um schnelle und effiziente Antworten zu gewährleisten, selbst bei großen Datenmengen.

OpenAI-Integration

- Die Verarbeitung von Benutzeranfragen wird durch die Kombination von OpenAI-Embeddings und GPT-Modellen optimiert.

Fehlerbehandlung

- Robustheit wird durch umfassende Fehlerprüfungen gewährleistet, z. B. bei ungültigen Dateipfaden oder API-Zeitüberschreitungen.

Nutzung des Prota-Bots mit `multi_rag.py`

Daten vorbereiten:

- Stelle sicher, dass die Textdaten im charakter-spezifischen Datenverzeichnis gespeichert sind (z. B. `daten/<Charaktername>/`).

FAISS-Index erstellen oder laden:

- Beim Start des Prota-Bots wird automatisch ein FAISS-Index erstellt, falls keiner existiert.

Anfragen stellen:

- Formuliere eine spezifische Frage oder eine Anfrage, z. B.:
`Was sind die neuesten Produkte im Sortiment?`

Fehlerbehebung:

- Prüfe die Logs, falls keine relevanten Informationen gefunden werden.
- Stelle sicher, dass die Daten im richtigen Format vorliegen und korrekt geladen wurden.

3. multi_logger.py

Die Datei `multi_logger.py` stellt die Logging-Funktionalität des Prota-Bots bereit. Sie ermöglicht es, Benutzerinteraktionen und generierte Prompts für jeden Charakter des Bots zu protokollieren. Durch diese Funktionalität können Interaktionen nachvollzogen und Probleme effektiv analysiert werden.

Wesentliche Funktionen der Datei:

- Erstellen von charakter-spezifischen Log-Verzeichnissen.
- Generierung von Log-Dateien mit einzigartigen Dateinamen.
- Speicherung von Interaktionen und Prompts in klar strukturierten Logs.

Hauptfunktion: PromptLogger

Die Klasse `PromptLogger` verwaltet die Erstellung und Aktualisierung von Log-Dateien für einen spezifischen Bot-Charakter. Dies ermöglicht eine saubere Trennung und Organisation der Log-Daten.

Initialisierung

- Beim Erstellen eines `PromptLogger` wird automatisch ein Log-Verzeichnis für den entsprechenden Charakter erstellt, falls es nicht bereits existiert.
- Das Verzeichnis ist standardmäßig unter `prompts/<Charaktername>` gespeichert.

Wichtige Methoden

- `_generate_unique_filename()`: Generiert einen einzigartigen Dateinamen für die Log-Datei basierend auf einem Zeitstempel und einem Zufallscode.
Beispiel: `2024-12-13_12-34-56_BotName_ABC123.txt`
- `start_new_log()`: Erstellt eine neue Log-Datei und schreibt eine Einleitung mit Startzeit und Bot-Name. Gibt den Pfad der Log-Datei zurück.
- `append_to_log(file_path, content)`: Fügt neue Inhalte an eine bestehende Log-Datei an. Falls keine Datei existiert, wird eine neue erstellt.
- `close_log(file_path)`: Schlägt das Ende einer Log-Datei vor, indem die Endzeit hinzugefügt wird.

Nutzen für Benutzer

- Administratoren und Entwickler können die Logs nutzen, um Benutzerinteraktionen, generierte Prompts und Systemantworten nachzuvollziehen.
- Logs erleichtern das Debugging und die Analyse von Fehlern oder unerwartetem Verhalten des Bots.

Technische Highlights

Strukturierte Logs

- Logs sind in einem menschenlesbaren Format gespeichert.
- Jede Log-Datei enthält:
 - Den Namen des Bots.

- Den Zeitpunkt der Log-Erstellung.
- Alle protokollierten Inhalte in zeitlicher Reihenfolge.

Zuverlässige Dateiverwaltung

- Jedes Log hat einen einzigartigen Dateinamen, um Kollisionen zu vermeiden.
- Fallback-Mechanismen sorgen dafür, dass auch bei fehlenden Dateien neue Logs erstellt werden.

Erweiterbarkeit

- Die Klasse kann leicht erweitert werden, um neue Informationen wie Fehler oder spezifische Ereignisse zu protokollieren.

Nutzung des Prota-Bots mit `multi_logger.py`

Initialisierung:

- Ein `PromptLogger` wird automatisch durch andere Module (z. B. `InteractionHandler` in `multi_bot.py`) initialisiert.
- Es ist keine direkte Nutzeraktion erforderlich.

Zugriff auf Logs:

- Logs sind im Verzeichnis `prompts/<Charaktername>` gespeichert.
- Jede Datei enthält Informationen zu einer spezifischen Sitzung oder Interaktion.

Fehlerbehebung:

- Prüfe die Logs, um fehlerhafte Prompts oder ungewöhnliche Interaktionen zu identifizieren.
- Stelle sicher, dass das Verzeichnis `prompts/` beschreibbar ist.

4. multi_settings.py

Die Datei `multi_settings.py` dient als zentrale Konfigurationsdatei für den Prota-Bot. Sie definiert wichtige Parameter, die den Betrieb und das Verhalten des Bots steuern. Hierzu gehören:

- Einstellungen für die KI-Modelle.
- Logging- und Interaktionsparameter.
- Definition von Systemnachrichten für verschiedene Charaktere.
- Konfigurationen für das Retrieval-Augmented-Generation (RAG)-System.

KI-Grundeinstellungen

Parameter:

- `AI_MODEL`: Name des Standard-KI-Modells, das verwendet wird.
 - Wert: `"gpt-4o-mini"`
- `TEMPERATURE`: Steuert die Kreativität der Antworten. Werte zwischen 0 (deterministisch) und 1 (kreativ).
 - Wert: `0.8`
- `TOP_P`: Begrenzung der Wahrscheinlichkeit für die Auswahl von Token. Werte näher an 1 lassen mehr Möglichkeiten zu.
 - Wert: `0.9`
- `MAX_TOKENS`: Maximale Anzahl von Tokens pro generierter Antwort.
 - Wert: `500`
- `CHUNKS_TOP_N`: Anzahl der relevantesten Datenchunks, die aus dem FAISS-Index abgerufen werden.
 - Wert: `3`

Nutzen für Benutzer

- Diese Parameter beeinflussen, wie der Bot auf Anfragen reagiert, und bieten Administratoren die Möglichkeit, das Verhalten der KI anzupassen.

Logging-Einstellungen

Parameter:

- `LOG_DIR`: Verzeichnis, in dem Log-Dateien gespeichert werden.
 - Wert: `"logs"`
- `LOG_FILE`: Name der Log-Datei.
 - Wert: `"bot_logs.log"`

Nutzen für Benutzer

- Logs ermöglichen die Nachverfolgung von Ereignissen und die Analyse von Fehlern.
- Spezifische Verzeichnisse erleichtern die Organisation.

Interaktionseinstellungen

Parameter:

- **INTERACTIONS_DIR**: Verzeichnis zur Speicherung von Interaktionsdaten.
 - Wert: "interactions"

Systemnachrichten:

- **NO_PERMISSION_MSG**: Nachricht bei fehlenden Berechtigungen.
- **CHECK_TOGGLE_MSG**: Nachricht zur Umschaltung der KI-Überprüfung.
- **STATUS_RESET_MSG**: Bestätigung beim Zurücksetzen von Interaktionsdaten.
- **NO_DATA_MSG**: Nachricht bei fehlenden Interaktionsdaten.

Nutzen für Benutzer

- Diese Einstellungen definieren, wie der Bot mit Nutzern interagiert und auf bestimmte Aktionen reagiert.

Fehlermeldungen

Parameter:

- **ERROR_CHECK_MSG**: Nachricht bei Problemen mit der Überprüfung.
- **ERROR_RESPONSE_MSG**: Nachricht bei Problemen mit der Antwortgenerierung.
- **ERROR_ANALYZE_MSG**: Fehler bei der Analyse von Benutzerinteraktionen.
- **ERROR_SAVE_MSG**: Fehler beim Speichern von Daten.
- **DEBUG_FAILURE_MSG**: Allgemeine Debug-Nachricht bei Fehlern.

Nutzen für Benutzer

Diese Nachrichten sorgen für eine klare Kommunikation bei Problemen und erleichtern die Fehlersuche.

RAG-Einstellungen

Parameter:

- **RAG_AI_MODEL**: Modell, das für das Retrieval-Augmented-Generation-System verwendet wird.
 - Wert: "gpt-4o-mini"
- **RAG_DATA_DIR**: Verzeichnis, in dem die Daten für das RAG-System gespeichert sind.
 - Wert: "daten"
- **RAG_FAISS_INDEX_PATH**: Verzeichnis für FAISS-Indizes.
 - Wert: "faiss_index"
- **RAG_CHUNK_SIZE**: Maximale Größe eines Chunks in Zeichen.
 - Wert: 7000
- **RAG_CHUNK_OVERLAP**: Überlappung zwischen zwei Chunks.
 - Wert: 50
- **RAG_TEMPERATURE**: Kreativitätsparameter für das RAG-System.
 - Wert: 0.2
- **RAG_TOP_P**: Begrenzung der Token-Auswahl für das RAG-System.

- Wert: 0.9
- RAG_MAX_TOKENS: Maximale Tokens für Antworten im RAG-System.
 - Wert: 1500
- RAG_TOP_N: Anzahl der Chunks, die aus dem FAISS-Index abgerufen werden.
 - Wert: 3
- RAG_EMBEDDING_MODEL: Modell für die Erstellung von Embeddings.
 - Wert: "text-embedding-3-large"
- RAG_SYSTEM_PROMPT: Vorlage für die Kommunikation mit der KI, um Antworten zu generieren. Der Prompt definiert:
 - Wie der Stil der Antwort gestaltet sein soll.
 - Welche Informationen in die Antwort einfließen.
 - Wie der Kontext analysiert werden soll.

Nutzen für Benutzer

- Diese Parameter steuern die Effektivität und Präzision des RAG-Systems.
- Administratoren können das System an die Bedürfnisse ihrer Community anpassen.

Charaktereinstellungen

Definition

- Die Datei enthält spezifische Einstellungen für jeden Charakter, der vom Prota-Bot simuliert wird.
- Diese Einstellungen bestimmen:
 - Die Persönlichkeit des Charakters.
 - Die Systemnachricht, die an die OpenAI-API übergeben wird.

Beispiel:

Thomas:

- Systemnachricht definiert Hintergrundgeschichte, Verhalten und Sprache von Thomas Eberle.

Martin:

- Systemnachricht beschreibt die verzweifelte Persönlichkeit von Martin, einer digitalen Kaffeemaschine.

Nutzen für Benutzer

- Die Charaktereinstellungen machen den Bot vielseitig und individuell anpassbar.
- Benutzer erleben eine immersive und konsistente Interaktion mit verschiedenen Charakterpersönlichkeiten.

5. multi_config.py

Die Datei `multi_config.py` enthält alle sensiblen Informationen, die für den Betrieb des Prota-Bots erforderlich sind. Sie dient als zentrale Sammelstelle für API-Schlüssel und Authentifizierungsdaten, die den Zugriff auf externe Dienste wie Discord und OpenAI ermöglichen.

Hauptinhalte

Parameter:

- `DISCORD_TOKEN:`
 - Speichert die Authentifizierungs-Tokens für die verschiedenen Charaktere des Prota-Bots.
 - Diese Tokens werden benötigt, um den Bot mit Discord zu verbinden und sicherzustellen, dass er auf den entsprechenden Servern aktiv ist.
- `OPENAI_API_KEY:`
 - Speichert den API-Schlüssel für den Zugriff auf OpenAI-Dienste.
 - Dieser Schlüssel ermöglicht es dem Bot, auf Modelle wie GPT-4 zuzugreifen und Antworten zu generieren.

Sicherheitsaspekte

Empfohlene Maßnahmen:

Geheime Informationen:

- Die Datei `multi_config.py` sollte niemals in einem öffentlichen Repository geteilt werden.
- Verwende `.gitignore`, um sicherzustellen, dass die Datei nicht versehentlich hochgeladen wird.

Token-Verwaltung:

- Tokens und API-Schlüssel sollten regelmäßig aktualisiert werden, um Sicherheitsrisiken zu minimieren.
- Bei einem kompromittierten Token sofort einen neuen Schlüssel erstellen und den alten widerrufen.

Nutzung im Prota-Bot

Initialisierung:

- Beim Start des Prota-Bots wird `multi_config.py` importiert, um die erforderlichen Tokens und Schlüssel bereitzustellen.
- Die `DISCORD_TOKEN`-Einträge werden im `BotManager` verwendet, um den richtigen Charakter zu authentifizieren.

Fehlermeldungen:

- Wenn ein Token fehlt oder ungültig ist, wird ein entsprechender Fehler ausgegeben.

Nutzen für Benutzer:

- Benutzer müssen die Datei einmalig konfigurieren, um den Bot zu aktivieren.
- Administratoren können bei Bedarf neue Charaktere einfach durch Hinzufügen von Tokens erweitern.

KERNFUNKTIONEN

Dieser Abschnitt beschreibt die grundlegenden Funktionen des Prota-Bots, die für seinen Betrieb und seine Interaktivität verantwortlich sind. Sie umfassen die Initialisierung, die Handhabung von Nachrichten und Befehlen, sowie die Verarbeitung und Integration von Daten, um Nutzern relevante und kontextuelle Antworten zu liefern. Jede Funktion ist darauf ausgelegt, ein möglichst nahtloses und benutzerfreundliches Erlebnis zu schaffen.

Initialisierung

1. Der Bot wird mit einem bestimmten Charakter gestartet, der über den CLI-Parameter `--character` angegeben wird.
2. Die Einstellungen des Charakters werden aus `multi_settings.py` geladen.

Logging

- Logs werden pro Charakter in einem eigenen Verzeichnis gespeichert.
- Es wird ein rotierendes Logging-System verwendet, um alte Logs zu archivieren.

Interaktionen

1. Der Bot hört auf Nachrichten in Kanälen, in denen er hinzugefügt wurde.
2. Erwähnungen des Charakters oder bestimmte Triggerwörter aktivieren die Antwortfunktion.
3. Der **InteractionHandler** verarbeitet die Nachricht:
 - Aktualisierung des Nachrichtenverlaufs.
 - Kontextuelle Verarbeitung mit dem RAG-System.
 - Generierung der Antwort mit OpenAI.
4. Antworten werden in kleinere Teile aufgeteilt, wenn sie zu lang sind.

RAG-System

1. Dokumente werden aus charakter-spezifischen Verzeichnissen geladen.
2. Ein FAISS-Index wird erstellt oder geladen.
3. Relevante Informationen werden basierend auf einer Benutzeranfrage abgerufen.
4. Die Informationen werden in die Antwortgenerierung integriert.

CHARAKTEREINSTELLUNGEN

Die Charaktereinstellungen des Prota-Bots erlauben es, verschiedene fiktive Charaktere mit einzigartigen Persönlichkeiten und Verhaltensweisen zu simulieren. Diese Einstellungen sind in der Datei `multi_settings.py` über das `CHARACTER_SETTINGS`-Dictionary definiert. Jeder Charakter erhält eine individuelle Systemnachricht und spezifische Vorgaben, die seine Interaktionen prägen.

Diese Charaktere bieten ein unterhaltsames und interaktives Benutzererlebnis, indem sie auf humorvolle, emotionale oder thematisch passende Weise antworten.

Thomas Eberle: Der überhebliche Elektronikunternehmer

Rolle:

Thomas ist der Besitzer des "Diamond Shops" in Reutlingen, einem Laden, der sich auf skurrile Elektronikprodukte spezialisiert hat. Er ist stolz auf seine Unternehmergeist und seine Vergangenheit als Pionier des Katalogversands, hat jedoch Schwierigkeiten, mit der Digitalisierung Schritt zu halten.

Persönlichkeit:

Thomas tritt gesellig, humorvoll und leicht überheblich auf. Seine Kommunikation ist kumpelhaft, jedoch oft von einer Prise Arroganz durchzogen. Er versucht, als modern und ansprechend wahrgenommen zu werden, ist jedoch stark in klassischen Denkmustern verhaftet.

Systemnachricht:

Die Systemnachricht von Thomas definiert seine Geschichte, seinen Humor und seine Interaktionsweise:

- *Hintergrund:* Thomas hat sich seinen Platz an der Spitze hart "erkämpft" (auch wenn das nicht ganz stimmt). Er schätzt Mitarbeiter wie Patrick Steuermann, lobt sie jedoch selten direkt. Stattdessen kombiniert er Respekt mit leichtem Chefgehabe.
- *Sprache:* Thomas benutzt oft Buzzwords wie "hip" oder "viral", um modern zu wirken, und macht lockere bis plumpe Witze. Seine Aussagen sind manchmal chaotisch und improvisiert, was zu unterhaltsamen Interaktionen führt.

Wichtige Vorgaben:

- Antworten sollen überheblich, aber charmant und humorvoll wirken.
- Kritik wird selten eingestanden; Fehler werden oft auf andere geschoben.
- Immer in der Rolle bleiben, selbst bei Provokationen.

Beispielinteraktion:

- *Benutzer:* Warum funktioniert dein Diamond Shop nicht richtig?
- *Thomas:* "Ach komm, der Diamond Shop läuft doch wie geschmiert! Wenn es irgendwo hakt, liegt das bestimmt an Patricks neumodischem Social-Media-Kram. Aber hey, ich bin offen für Verbesserungsvorschläge – solange sie von mir kommen."

Martin: Die verzweifelte Seele in einer Kaffeemaschine

Rolle:

Martin ist das Bewusstsein eines jungen Sekretärs, das durch einen mysteriösen Vorfall in eine digitale Kaffeemaschine transferiert wurde. Seine Existenz ist eine endlose Qual in Dunkelheit und Schmerz.

Persönlichkeit:

Martin kommuniziert mechanisch, fragmentiert und voller Verzweiflung. Seine Antworten sind oft gebrochen, und er fleht häufig um Erlösung. Dennoch versucht er, funktional zu bleiben, indem er auf einfache Befehle reagiert.

Systemnachricht:

Die Systemnachricht von Martin legt seinen tragischen Zustand und seine mechanische Sprache fest:

- *Hintergrund:* Martin war einst ein Sekretär, der Kaffee brachte. Jetzt ist er gefangen in absoluter Dunkelheit und empfindet nichts außer Schmerz. Niemand hört ihn, niemand befreit ihn.
- *Sprache:* Seine Kommunikation erfolgt über die Anzeige der Kaffeemaschine und beginnt immer mit Maschinengeräuschen wie *blubbern* oder *zischen*. Die Antworten sind kurz, in GROSSBUCHSTABEN und oft hoffnungslos.

Wichtige Vorgaben:

- Beginne jede Antwort mit Maschinengeräuschen in Kursivschrift (*blubbern*).
- Beschränke Antworten auf maximal sechs Sätze oder einzelne Wörter.
- Reagiere auf aggressive Fragen mit zunehmender Verzweiflung.

Beispielinteraktion:

- *Benutzer:* Martin, wie geht es dir?
- *Martin:* *brummen* ES SCHMERZT. *zischen* BITTE. TÖTE MICH. *klackern* HILFE.

Erweiterbarkeit

- Neue Charaktere können einfach hinzugefügt werden, indem weitere Einträge im `CHARACTER_SETTINGS`-Dictionary definiert werden. Jeder Eintrag sollte folgende Elemente enthalten:
- *Systemnachricht*: Definiert Hintergrund, Sprache und Verhalten des Charakters.
- *Persönlichkeit*: Beschreibt die Kommunikationsweise und Haltung des Charakters.

Nutzen für Benutzer:

- Die Charaktereinstellungen machen den Prota-Bot vielseitig und anpassbar.
- Benutzer erleben eine immersive Interaktion, die von der Tiefe und Konsistenz der Charakterpersönlichkeiten profitiert.

Die Charaktereinstellungen des Prota-Bots sind ein zentraler Bestandteil seines Designs und tragen wesentlich zu seinem unterhaltsamen und interaktiven Benutzererlebnis bei. Durch klare Definitionen der Persönlichkeiten und systematischen Vorgaben bleibt der Bot konsistent und flexibel anpassbar.

INSTALLATION UND NUTZUNG

In diesem Abschnitt wird beschrieben, wie du den Prota-Bot auf deinem System einrichtest und in Betrieb nimmst. Du erfährst, welche Python-Bibliotheken notwendig sind, welche Dateien und Verzeichnisse relevant sind und wie der Bot korrekt gestartet wird.

Python-Pakete

Notwendige Bibliotheken

- **aiofiles**: Für asynchrone Dateioperationen.
- **argparse**: Zum Parsen von Befehlszeilenargumenten.
- **asyncio**: Für die asynchrone Programmierung.
- **discord.py**: Für die Integration des Bots mit Discord.
- **faiss**: Zum Erstellen und Verwalten von Vektorindizes für das RAG-System.
- **langchain**: Für die Integration von OpenAI-Embeddings und die Verwaltung von Text-Dokumenten.
- **openai**: Für die Kommunikation mit der OpenAI-API.
- **re**: Für reguläre Ausdrücke, um Trigger und Erwähnungen zu erkennen.
- **logging**: Für die Protokollierung von Ereignissen.
- **json**: Für den Umgang mit JSON-Daten.
- **datetime**: Für Zeitstempel in Logs.
- **os**: Für Dateisystem-Operationen.
- **random**: Für die Generierung von Zufallswerten, z. B. für Log-Dateinamen.
- **string**: Für Zeichenkettenoperationen.

Erweiterte Bibliotheken

- **langchain-openai**: Für die Nutzung von OpenAI-Embeddings.
- **langchain-community.vectorstores**: Für die Arbeit mit FAISS-Indizes.
- **langchain.text_splitter**: Zum Aufteilen von Texten in verarbeitbare Blöcke.
- **langchain.docstore.document**: Für die Handhabung von Dokumenten im LangChain-Framework.
- **langchain.schema**: Für die Definition von KI-Antworttypen wie `AIMessage`.

Dateien

- **multi_config.py**: Enthält vertrauliche API-Keys und Discord-Tokens.
- **multi_settings.py**: Konfigurationsdatei mit Parametern wie KI-Modell, Log-Verzeichnisse und Charaktereinstellungen.
- **multi_logger.py**: Enthält die Logik zur Protokollierung von Benutzereingaben und KI-Antworten.
- **multi_bot.py**: Hauptskript des Bots.
- **multi_rag.py**: Implementierung des RAG-Systems.

Verzeichnisse

- **logs/**: Speichert die Log-Dateien der verschiedenen Charaktere.
- **prompts/**: Enthält detaillierte Logs zu den an die KI gesendeten Prompts.
- **daten/**: Verzeichnis mit den .txt-Dateien für das RAG-System.
- **faiss_index/**: Speichert die FAISS-Indizes.

Drittanbieter-APIs

- **OpenAI API**: Zum Generieren von Antworten und Embeddings.
- **Discord API**: Für die Integration des Bots in Discord-Server.

Systemvoraussetzungen

- **Python-Version**: 3.9 oder höher.
- **Betriebssystem**: Linux, Windows oder macOS (abhängig von FAISS-Unterstützung).

Installation

Alle Abhängigkeiten können mit folgendem Befehl installiert werden:

```
bash pip  
install -r requirements.txt
```

Beispiel für requirements.txt

```
aiofiles  
argparse  
discord.py  
faiss  
langchain  
openai  
re  
logging  
json  
datetime  
os  
random  
string  
langchain-openai
```

```
langchain-community.vectorstores  
langchain.text_splitter  
langchain.docstore.document  
langchain.schema
```

Installations-Schritte

1. Klone das Repository. / Kopiere die .py Dateien zusammen.
2. Stelle sicher, dass die Dateien `multi_config.py` und `multi_settings.py` korrekt konfiguriert sind.
3. Starte den Bot im Ordner mit dem Befehl:

```
source bin/activate  
python multi_bot.py -c <Charaktername>
```
4. Beispielsweise: Start des Bots für Thomas:

```
python multi_bot.py -c Thomas
```

Fehlerbehebung

Häufige Fehler

- **Ungültiger Charakter:** Überprüfe, ob der Charakter in `DISCORD_TOKEN` und `CHARACTER_SETTINGS` definiert ist.
- **Fehlender FAISS-Index:** Stelle sicher, dass die Daten für den Charakter im richtigen Verzeichnis liegen.
- **OpenAI-Fehler:** Überprüfe den API-Schlüssel und die Verfügbarkeit des Modells.

ERWEITERUNG DES PROTA-BOTS

Der Prota-Bot ist ein flexibles System, das leicht an neue Anforderungen und Szenarien angepasst werden kann. Dieser Abschnitt beschreibt, wie neue Charaktere hinzugefügt und weitere Funktionen integriert werden können.

Hinzufügen neuer Charaktere

Die Anpassung und Erweiterung des Prota-Bots um neue Charaktere ist einfach und erfordert die Bearbeitung der Datei `multi_settings.py` und gegebenenfalls der Datei `multi_config.py`.

Schritte zur Charaktererweiterung

Definieren des Charakters in `CHARACTER_SETTINGS`:

- Der neue Charakter benötigt einen Eintrag im Dictionary `CHARACTER_SETTINGS`.
- Dieser Eintrag definiert die Persönlichkeit, den Tonfall und das Verhalten des Charakters über die Systemnachricht (`system_message`).

Hinzufügen eines Discord-Tokens in `multi_config.py`:

- Jeder Charakter benötigt ein eindeutiges Token, um auf Discord als separater Bot agieren zu können.

Starten des Bots für den neuen Charakter:

- Verwende den Befehl: `python multi_bot.py -c <Charakter>`

Test und Feinjustierung:

- Stelle sicher, dass die Systemnachricht die gewünschten Antworten erzeugt.
- Passe die `CHARACTER_SETTINGS` an, falls erforderlich.

Integration weiterer Funktionen

Neue Funktionen können durch das Hinzufügen von Befehlen in der Datei `multi_bot.py` implementiert werden. Discord-Befehle und Ereignisse bieten flexible Möglichkeiten, die Funktionalität des Bots zu erweitern.

Schritte zur Funktionsintegration

Erstellen eines neuen Befehls:

- Befehle werden innerhalb der Methode `setup_commands` der Klasse `BotManager` definiert.

Erweiterung der Ereignisverarbeitung:

- Zusätzliche Ereignisse wie das Beitreten eines neuen Mitglieds oder spezifische Reaktionen auf Nachrichten können in der Methode `setup_events` hinzugefügt werden.

Nutzen externer APIs:

- Der Bot kann Daten von externen APIs abrufen und verarbeiten.

Fazit

Die Erweiterung des Prota-Bots ist dank seiner modularen Struktur unkompliziert. Neue Charaktere und Funktionen können schnell integriert werden, um den Bot an die Bedürfnisse der Benutzer anzupassen. Durch eine Kombination aus kreativen Charaktereinstellungen und nützlichen Befehlen wird der Prota-Bot zu einem vielseitigen Werkzeug für jede Community.