Sync Lab S.r.l.

Progetto di stage

Andrea Cecchin

Proof of Concept

Report attività di analisi, progettazione e codifica


SYNCRAG

Indice

Introduzione

1.1 Scopo del documento

Il seguente documento intende esporre le attività, svolte durante il periodo di stage relativo all'implementazione delle tecnologie per la Retrieval-Augmented Generation, di realizzazione di un prototipo integrante le tecnologie per la RAG in un backend basato su Java Spring, con un frontend TypeScript con React.

1.2 Riferimenti informativi

- Scheda tecnica di Phi-3 su Ollama:
  `https://ollama.com/library/phi3`
- Paper ufficiale di Phi-3:
  `https://arxiv.org/abs/2404.14219`
- Paper ufficiale di Gemini 1.5:
  `https://arxiv.org/pdf/2403.05530`
- Paper ufficiale di PaLM 2:
  `https://arxiv.org/pdf/2305.10403`
- Sito web di Open AI (modello GPT 3.5):
  `https://platform.openai.com/docs/models/gpt-3-5-turbo`
- Scheda tecnica di All-MiniLM-L6-v2:
  `https://www.sbert.net/docs/pretrained_models.html`
- Sito web di LangChain4J:
  `https://docs.langchain4j.dev`
- Documentazione di LangChain4J:
  `https://docs.langchain4j.dev/apidocs/index.html`
- Repository Github di LangChain4J:
  `https://github.com/langchain4j/langchain4j`

- Sito web di Spring:
  `https://spring.io`
- Sito web di React:
  `https://react.dev/`
- Sito web di H2:
  `https://www.h2database.com/html/main.html`
- Bcrypt - Wikipedia:
  `https://it.wikipedia.org/wiki/Bcrypt`

Analisi dei Requisiti

Le attività di realizzazione del prototipo finale del progetto di stage sono incominciate dall'analisi dei requisiti. In particolare, è stato necessario individuare quali sono gli attori coinvolti nell'interazione con il prodotto.
Una volta individuati gli attori, sono stati analizzati quali sono gli Use Case principali che sono presenti in un ipotetico prodotto che permette la RAG su un insieme di documenti, così da individuare i requisiti che il Proof of Concept deve soddisfare.

2.1 Attori

A seguito dell'analisi delle tipologie di utenti che si relazionerebbero con un prodotto finale, sono stati individuati due diversi attori.

Utente generico: Rappresenta un utente non autenticato, che ha accesso a una parte limitata delle operazioni chiave nella Retrieval-Augmented Generaion. In particolare, questo attore può accedere alle funzionalità di chat, con cui interrogare i modelli, ma non ha la possibilità di visualizzare o modificare la lista dei documenti presenti nel database vettoriale.

Admin: Rappresenta un utente che ha effettuato l'autenticazione e ha ricevuto l'accesso a tutte le funzionalità del prodotto. In aggiunta a quanto può già fare l'utente generico, l'amministratore è in grado di visualizzare la lista dei documenti presenti nel database vettoriale, oltre a poterne aggiungere o rimuovere.

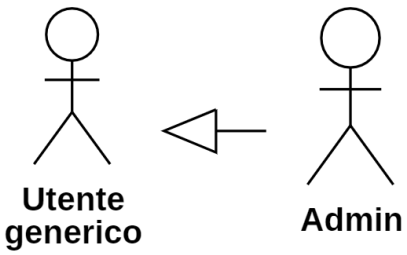

Figura 1: Attori individuati

2.2 Casi d'uso

UC1 - Visualizza modelli

Descrizione: L'utente vuole visualizzare la lista dei modelli supportati dal sistema nella generazione delle risposte.

Attore primario: Utente generico.

Pre condizioni: Il sistema è inizializzato.

Post condizioni: Il sistema mostra la lista dei modelli supportati.

Scenario:

1. L'utente visualizza la lista dei modelli supportati.

UC2 - Seleziona modello

Descrizione: L'utente vuole selezionare il modello dal quale ricevere le risposte alle proprie domande.

Attore primario: Utente generico.

Pre condizioni: L'utente ha visualizzato la lista dei modelli supportati.

Post condizioni: Il sistema utilizzerà il modello selezionato per generare le risposte.

Scenario:

1. L'utente visualizza la lista dei modelli supportati (**UC1**);
2. L'utente seleziona il modello da utilizzare nella generazione delle risposte.

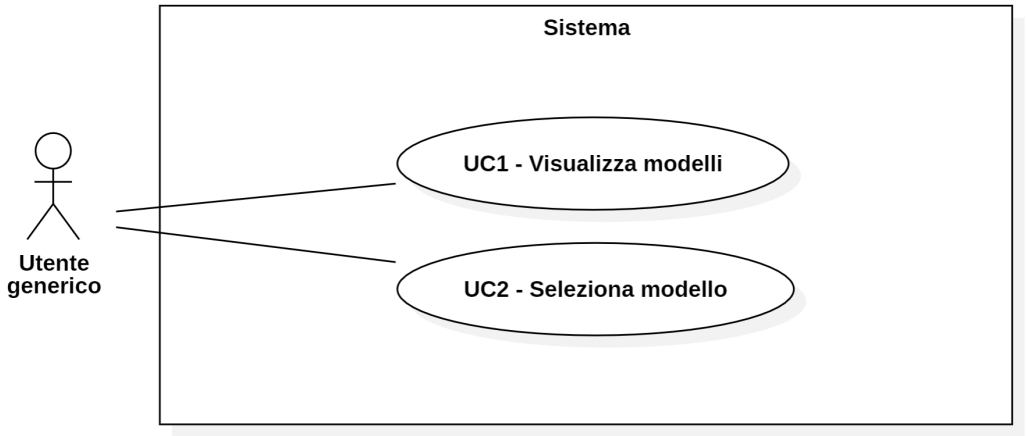

Figura 2: UML degli use cases UC1 e UC2

UC3 - Inserisci domanda

Descrizione: L'utente vuole inviare una domanda al chatbot, il quale dovrà rispondere utilizzando il modello desiderato.

Attore primario: Utente generico.

Pre condizioni: L'utente ha selezionato il modello da interrogare.

Post condizioni: Il sistema riceve la domanda posta dall'utente.

Scenario:

1. L'utente visualizza la lista dei modelli supportati (**UC1**);
2. L'utente seleziona il modello da utilizzare nella generazione delle risposte (**UC2**);
3. L'utente digita la domanda e la invia al sistema.

UC4 - Visualizza risposta

Descrizione: L'utente vuole visualizzare la risposta del chatbot alla domanda che ha posto in precedenza.

Attore primario: Utente generico.

Pre condizioni: L'utente ha inviato una domanda al chatbot.

Post condizioni: Il sistema mostra la risposta elaborata dal sistema.

Scenario:

1. L'utente digita la domanda e la invia al sistema (**UC3**);
2. L'utente visualizza la risposta ricevuta.

Generalizzazioni:

1. Visualizza testo della risposta (**UC4.1**);
2. Visualizza autore della risposta (**UC4.2**).

UC4.1 - Visualizza testo della risposta

Descrizione: L'utente vuole visualizzare il messaggio di riposta del chatbot alla domanda che ha posto in precedenza.

Attore primario: Utente generico.

Pre condizioni: L'utente ha inviato una domanda al chatbot.

Post condizioni: Il sistema mostra il messaggio di risposta elaborata dal sistema.

Scenario:

1. L'utente digita la domanda e la invia al sistema (**UC3**);
2. L'utente visualizza il messaggio di risposta ricevuto.

UC4.2 - Visualizza autore della risposta

Descrizione: L'utente vuole poter riconoscere il modello che ha generato la risposta.

Attore primario: Utente generico.

Pre condizioni: L'utente ha inviato una domanda al chatbot.

Post condizioni: Il sistema mostra la risposta elaborata dal sistema, e con essa il modello che l'ha generata.

Scenario:

1. L'utente digita la domanda e la invia al sistema (**UC3**);
2. L'utente visualizza l'autore della risposta ricevuta.

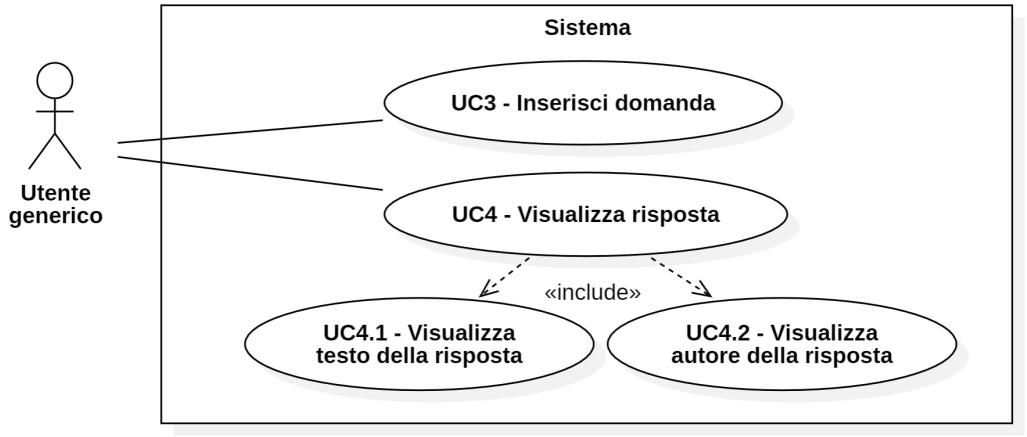

Figura 3: UML degli use cases UC3 e UC4

UC5 - Autenticazione

Descrizione: L'utente vuole autenticarsi per accedere alle funzionalità dell'amministratore.

Attore primario: Utente generico.

Pre condizioni: Il sistema è inizializzato correttamente.

Post condizioni: L'utente generico è ora autenticato come amministratore.

Scenario:

1. L'utente effettua l'autenticazione come amministratore.

Generalizzazioni:

1. Inserimento password (**UC5.1**).

Scenario alternativo: Visualizza errore nell'autenticazione (**UC5.1**).

UC5.1 - Inserimento password

Descrizione: L'utente vuole inserire la password per autenticarsi come amministratore.

Attore primario: Utente generico.

Pre condizioni: Il sistema è inizializzato correttamente.

Post condizioni: L'utente ha inserito la password per l'autenticazione.

Scenario:

1. L'utente inserisce la password per l'autenticazione come amministratore.

UC6 - Visualizza errore nell'autenticazione

Descrizione: Il sistema mostra all'utente un messaggio di errore che lo avvisa che l'autenticazione non è andata a buon fine.

Attore primario: Utente generico.

Pre condizioni: L'utente inserisce una password errata e tenta di effettuare l'autenticazione.

Post condizioni: Il sistema non effettua l'autenticazione e l'utente non ha accesso alle funzionalità dell'amministratore.

Scenario:

1. L'utente inserisce una password errata (**UC5.1**) nel tentativo di accedere come amministratore (**UC5**);
2. Il sistema mostra un messaggio di errore.

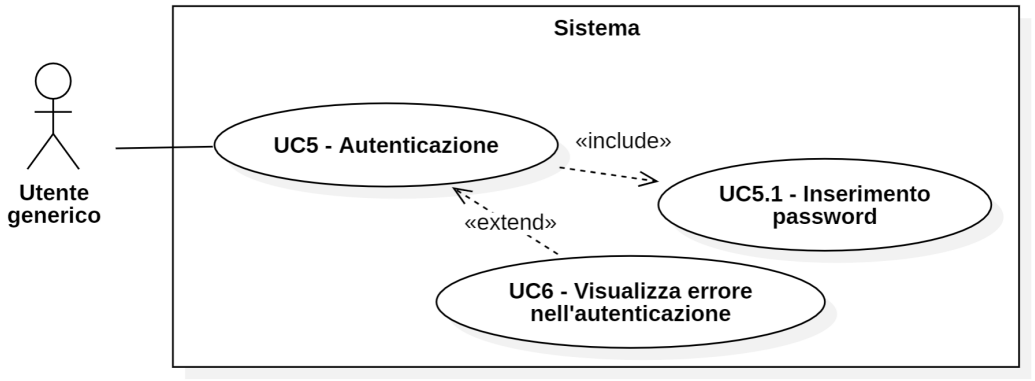

Figura 4: UML degli use cases UC5 e UC6

UC7 - Visualizza documenti

Descrizione: L'amministratore vuole visualizzare la lista dei documenti caricati nel database, sui quali è possibile fare delle domande al chatbot.

Attore primario: Admin.

Pre condizioni: L'utente ha effettuato l'autenticazione come amministratore.

Post condizioni: Il sistema mostra la lista dei documenti presenti nel database.

Scenario:

1. L'utente ha effettuato l'accesso come amministratore (**UC5**);
2. Il sistema mostra la lista dei documenti presenti nel database.

UC8 - Aggiungi documento

Descrizione: L'amministratore vuole aggiungere un nuovo documento nel database, su cui poter fare delle domande al chatbot.

Attore primario: Admin.

Pre condizioni: L'utente ha effettuato l'autenticazione come amministratore.

Post condizioni: Il sistema ha processato correttamente il nuovo documento e mostra la lista dei documenti aggiornata.

Scenario:

1. L'utente ha effettuato l'accesso come amministratore (**UC5**);
2. Il sistema mostra la lista di tutti i documenti presenti nel database (**UC7**);
3. L'amministratore inserisce il nuovo documento.

UC9 - Rimuovi documento

Descrizione: L'amministratore vuole rimuovere un documento nel database, così che il chatbot non fornisca più risposte su di esso.

Attore primario: Admin.

Pre condizioni: L'utente ha effettuato l'autenticazione come amministratore.

Post condizioni: Il sistema ha rimosso dal database il documento e mostra la lista dei documenti aggiornata.

Scenario:

1. L'utente ha effettuato l'accesso come amministratore (**UC5**);
2. Il sistema mostra la lista di tutti i documenti presenti nel database (**UC7**);
3. L'amministratore elimina uno dei documenti presenti dalla lista.

UC10 - Effettua log out

Descrizione: L'amministratore vuole effettuare il log out.

Attore primario: Admin.

Pre condizioni: L'utente ha effettuato l'autenticazione come amministratore.

Post condizioni: L'utente non è più autenticato e non ha più accesso alle funzionalità dell'amministratore.

Scenario:

1. L'utente ha effettuato l'accesso come amministratore (**UC5**);
2. L'amministratore effettua il log out.

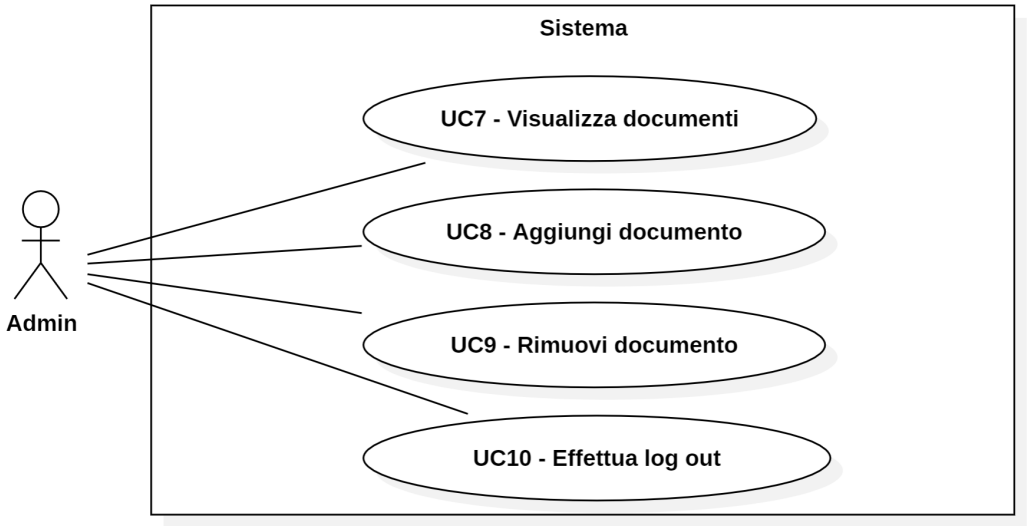

Figura 5: UML degli use cases UC7, UC8, UC9 e UC10

2.3 Requisiti

Requisiti funzionali

Codice	Descrizione	Classificazione	Fonte
RFZ-1	L'utente deve poter visualizzare la lista dei modelli supportati nell'interazione col chatbot.	Opzionale	UC1
RFZ-2	L'utente deve poter selezionare quale modello utilizzare nell'interazione col chatbot.	Opzionale	UC2
RFO-3	L'utente deve poter inviare delle domande al chatbot.	Obbligatorio	UC3
RFO-4	L'utente deve poter visualizzare la risposta data dal chatbot.	Obbligatorio	UC4.1
RFO-5	L'utente deve poter visualizzare il modello che ha generato la risposta data dal chatbot.	Obbligatorio	UC4.2

Codice	Descrizione	Classificazione	Fonte
RFO-6	L'utente deve poter effettuare l'autenticazione per accedere alle funzionalità dell'amministratore.	Obbligatorio	UC5
RFO-7	L'utente deve visualizzare un messaggio di errore quando l'autenticazione non va a buon fine.	Obbligatorio	UC6
RFO-8	L'amministratore deve poter visualizzare quali sono i documenti presenti nel database.	Obbligatorio	UC7
RFO-9	L'amministratore deve poter inserire un nuovo documento su cui poter interrogare il chatbot.	Obbligatorio	UC8
RFO-10	L'amministratore deve poter inserire rimuovere un documento dal database.	Obbligatorio	UC9
RFO-11	L'amministratore deve poter effettuare il log out.	Obbligatorio	UC10

Tabella 1: Requisiti funzionali individuati

Requisiti di vincolo

Codice	Descrizione	Classificazione	Fonte
RVO-1	Il backend del prototipo deve essere sviluppato in Java, utilizzando Spring.	Obbligatorio	Piano di Lavoro
RVO-2	Il frontend del prototipo deve essere sviluppato in TypeScript, utilizzzando React.	Obbligatorio	Piano di Lavoro
RVO-3	Il prototipo deve permettere l'utilizzo del modello Phi-3 mini.	Obbligatorio	Piano di Lavoro

Codice	Descrizione	Classificazione	Fonte
RVO-4	Il prototipo deve permettere l'utilizzo del modello Gemini 1.5 Pro.	Obbligatorio	Piano di Lavoro
RVD-5	Il prototipo deve permettere l'utilizzo del modello GPT 3.5 Turbo.	Desiderabile	Piano di Lavoro
RVZ-6	Il prototipo deve permettere l'utilizzo del modello PaLM 2 Bison.	Opzionale	Interna
RVZ-7	Il prototipo deve permettere la selezione in tempo reale del modello da interrogare.	Opzionale	Piano di Lavoro

Tabella 2: Requisiti di vincolo individuati

Progettazione e codifica

3.1 Progettazione interfaccia grafica

Considerando il prodotto dell'attività di analisi dei requisiti, il prototipo finale avrà due aree principali: la pagina relativa all'amministratore, con chatbot e area di gestione dei documenti a sistema, e la pagina per gli utenti generici, comprendente della sola chat.
Data la presenza di un meccanismo di autenticazione, per differenziare il ruolo e i privilegi degli utilizzatori del prodotto, è prevista una pagina apposita per il login.

Home Page

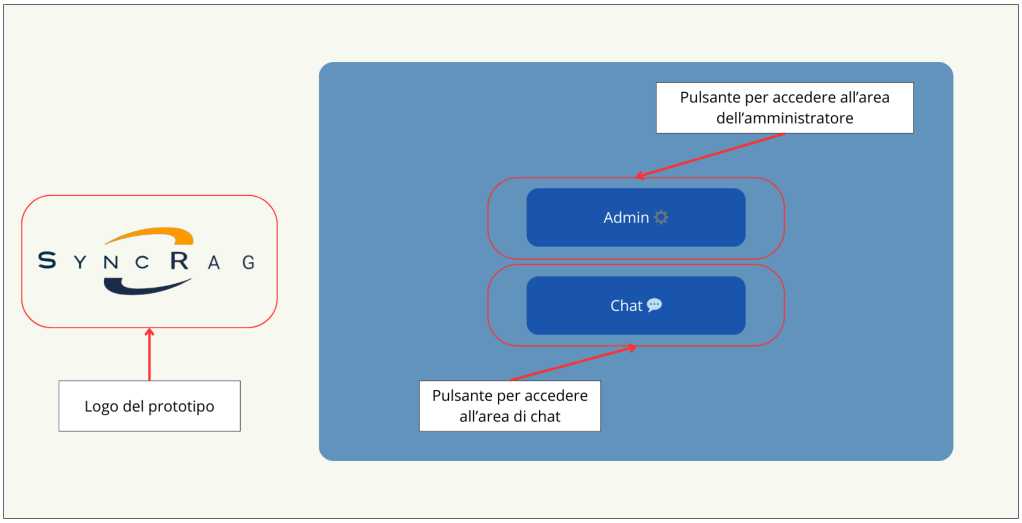

Figura 6: Progettazione GUI della Home Page

Elementi

- Logo del prototipo, che riprende il logo e il nome aziendale di Sync Lab, giocando sulla sigla di Retrieval-Augmented Generation.

- Pulsante per accedere all'area dell'amministratore. Quando premuto, esso rimanderà alla pagina dell'amministratore se ancora autenticato, altrimenti sarà mostrata la pagina per il login dell'utente.
- Pulsante per accedere all'area di chat: non necessita di autorizzazioni, infatti è aperta ad ogni utente generico.

Pagina di autenticazione

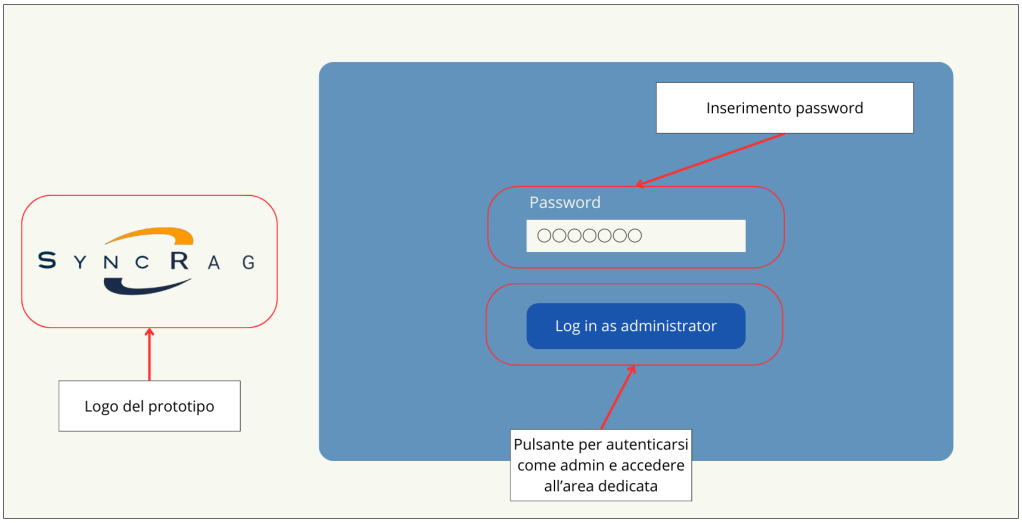

Figura 7: Progettazione GUI della pagina di autenticazione admin

Elementi

- Logo del prototipo, che riprende il logo e il nome aziendale di Sync Lab, giocando sulla sigla di Retrieval-Augmented Generation.
- Form su cui inserire la password.
- Pulsante per effettuare il login. In caso di successo dell'operazione, l'utente amministratore verrà correttamente indirizzato nella pagina relativa. In caso di errore nell'autenticazione (inserimento errato della password), sarà mostrato un messaggio di errore all'utente.

Pagina dell'amministratore

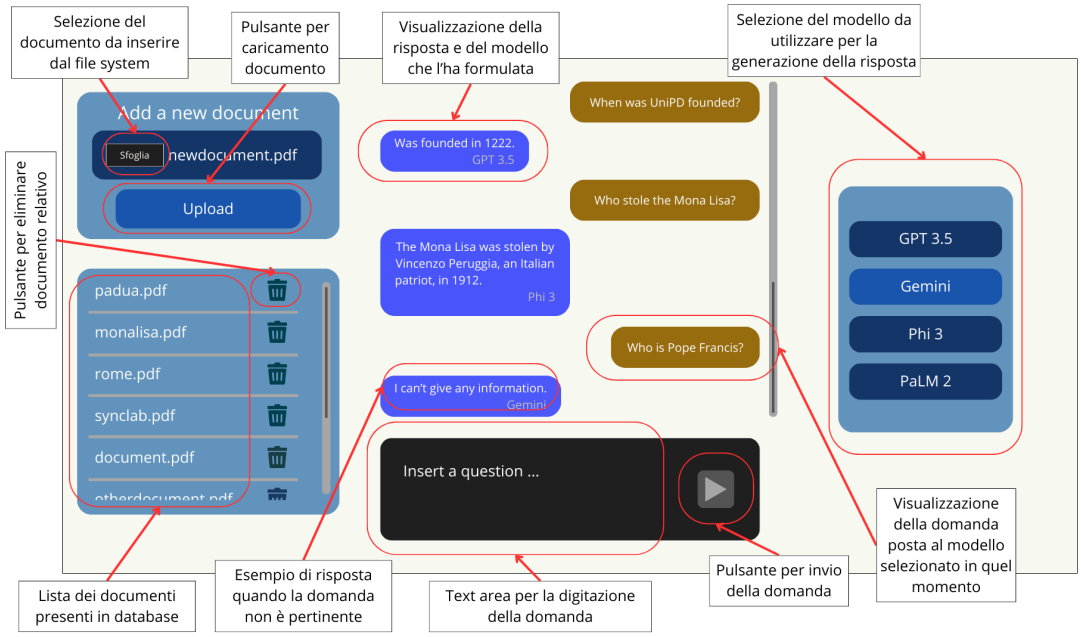

Figura 8: Progettazione GUI della pagina dell'admin

Elementi

- Area di gestione dei documenti a sistema, nella quale vengono mostrati i nomi dei documenti presenti. Per ogni documento in lista, è associato un bottone con cui richiedere la sua rimozione dal sistema.
- Form di inserimento di un nuovo documento.
- Pulsante per effettuare il caricamento del nuovo documento selezionato.
- Area di chat, nella quale vengono mostrati i messaggi scambiati tra utente e chatbot. L'autore del messaggio (utente o sistema) è identificato dal colore e dall'allineamento del singolo messaggio. Per ogni messaggio dato dal chatbot, oltre al testo della risposta è riportato il nome del modello che lo ha generato.
- Text area su cui digitare la domanda per il chatbot, e relativo pulsante per inviarla.
- Menù per selezionare, in tmepo reale, il modello che dovrà generare la risposta.

Pagina di chat

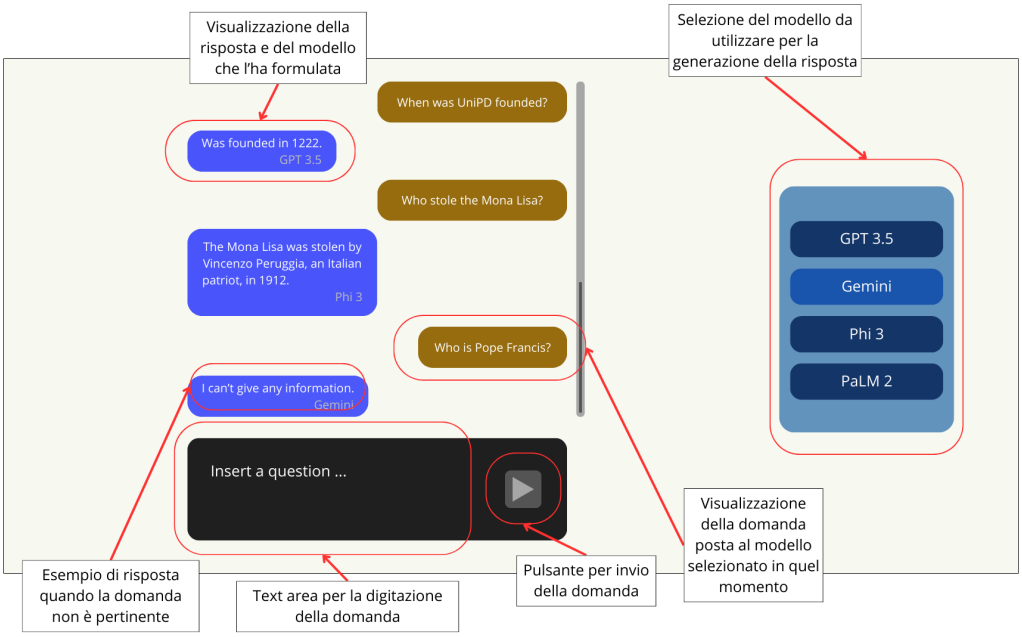

Figura 9: Progettazione GUI della pagina di chat

Elementi

- Area di chat, nella quale vengono mostrati i messaggi scambiati tra utente e chatbot. L'autore del messaggio (utente o sistema) è identificato dal colore e dall'allineamento del singolo messaggio. Per ogni messaggio dato dal chatbot, oltre al testo della risposta è riportato il nome del modello che lo ha generato.
- Text area su cui digitare la domanda per il chatbot, e relativo pulsante per inviarla.
- Menù per selezionare, in tempo reale, il modello che dovrà generare la risposta.

3.2 Specifica architetturale

Pattern architetturale

Il prototipo è stato sviluppato seguendo il modello della layered architecure. In particolare, sono stati individuati quattro strati:

Presentation Layer: contiene la logica di presentazione, ovvero come l'utente visualizza le informazioni e interagisce con il prodotto. Nel caso specifico di questo prototipo, lo strato di presentazione è costituito dal codice del frontend React.

Business Layer: contiene la logica di business del prodotto. Gestisce le chiamate dal Presentation Layer e compie tutte le funzionalità dell'applicativo, manipolando e utilizzando i dati presenti nei database e recuperati tramite il Persistence Layer. Nel caso specifico di questo prototipo, il frontend è in comunicazione con la logica di business del backend, presente nelle classi Service, grazie a un Controller.

Persistence Layer: contiene la logica di persistenza, incaricata di gestire le operazioni CRUD e quindi l'accesso ai dati dell'applicativo presenti nei database. Mette in comunicazione il Business Layer con le basi di dati, dato le informazioni necessarie per svolgere le funzionalità della logica di business.

Database Layer: contiene le basi di dati utilizzate dall'applicativo per memorizzare e utilizzare le informazioni necessarie al suo funzionamento. Nel caso specifico di questo prototipo, sono utilizzati due distinti database: uno per contenere in modo sicuro le informazioni di autenticazione dell'admin, l'altro per memorizzare i vettori di embedding e il testo relativo.

Endpoint

localhost:8080/api/ask

Descrizione: Richiesta perinterrogare un modello su una data domanda.
Metodo HTTP: GET
Parametri richiesti: Stringa della domanda, stringa che identifica il modello da interrogare.
Output atteso: Stringa della risposta.
Metodo Controller associato: String askQuestion(String question, String model)

localhost:8080/api/documents

Descrizione: Richiesta per recuperare il nome dei documenti presenti nel database vettoriale.
Metodo HTTP: GET
Parametri richiesti: Nessuno.
Output atteso: Lista di stringhe dei nome dei documenti in database.
Metodo Controller associato: List<String> getDocuments()

localhost:8080/api/documents

Descrizione: Richiesta per aggiungere un nuovo documento nel database vettoriale.
Metodo HTTP: POST
Parametri richiesti: Documento da processare ed inserire.
Output atteso: Stringa di conferma inserimento.
Metodo Controller associato: String addDocument(MultipartFile newDocument)

localhost:8080/api/documents

Descrizione: Richiesta di eliminazione di un documento presente nel database vettoriale.
Metodo HTTP: DELETE
Parametri richiesti: Stringa del nome del documento da rimuovere, nel formato ”nomedocumento.estensione”.
Output atteso: Stringa di conferma rimozione.
Metodo Controller associato: String deleteDocument(String toDelete)

localhost:8080/api/auth

Descrizione: Richiesta per effettuare l'autenticazione come admin.
Metodo HTTP: GET
Parametri richiesti: Stringa della password inserita.
Output atteso: Valore booleano che rappresenta la buona riuscita o meno dell'autenticazione.
Metodo Controller associato: Boolean auth(String password)

Classi del backend

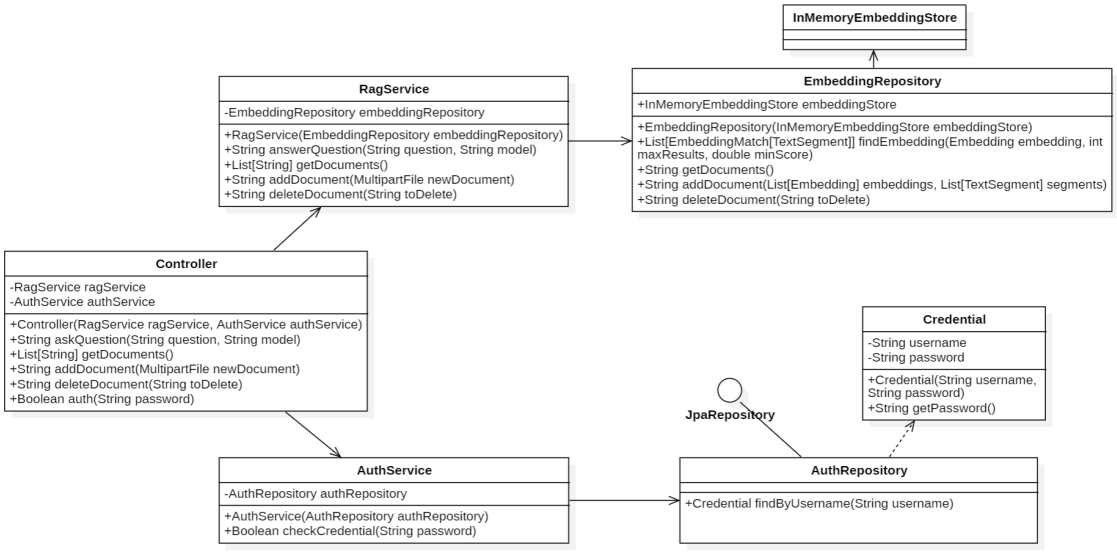

Figura 10: UML delle classi del backend

Controller

Descrizione: È la classe che collega frontend e backend del prototipo. Questa classe gestisce le chiamate HTTP provenienti dal client, mappandole a metodi specifici con cui accedere alla business logic dell'applicativo.

Attributi

- private final RagService ragService;
- private final AuthService authService.

Metodi

- public String askQuestion(String question, String model):
  metodo che riceve dal client la domanda posta dall'utente e il modello che deve generare la risposta, chiamando answerQuestion(String question, String model) del RagService che genererà la risposta.
  Raggiungibile con una chiamata GET dall'endpoint localhost:8080/api/ask.
- public List<String> getDocuments():
  chiama l'omonimo metodo del RagService, che recupererà i nomi dei documenti rpesenti nel database vettoriale.

Raggiungibile con una chiamata GET dall'endpoint localhost:8080/api/documents.

- public String addDocument(MultipartFile newDocument):
  metodo che riceve il file che l'utente vuole inserire nel database vettoriale, chiamando addDocument(MultipartFile newDocument) del RagService per processarlo.
  Raggiungibile con una chiamata POST dall'endpoint localhost:8080/api/documents.

- public String deleteDocument(String toDelete):
  metodo che riceve dal client, sotto forma di stringa, il nome del documento che deve essere rimosso dal database vettoriale e lo passa all'omonimo metodo del RagService.
  Raggiungibile con una chiamata DELETE dall'endpoint localhost:8080/api/documents.

- public Boolean auth(String password):
  metodo che riceve dal client la password inserita dall'utente nel tentativo di autenticarsi come amministratore. Chiama checkCredential(String password) di AuthService e restituisce, tramite valore booleano, il risultato dell'autenticazione.
  Raggiungibile con una chiamata GET dall'endpoint localhost:8080/api/auth.

RagService

Descrizione: È la classe che contiene la business logic per generare le risposte del chatbot interrogando i modelli, per processare e aggiungere nuovi documenti al database vettoriale, per visualizzare i documenti già presenti e per rimuoverne. Riceve le chiamate dal Controller, utilizza la classe EmbeddingRepository per ricevere i dati su cui operare e restituisce i risultati delle operazioni al Controller stesso.

Attributi

- private final EmbeddingRepository embeddingRepository.

Metodi

- public String answerQuestion(String question, String model):
  metodo responsabile di generare le risposte del chatbot. Utilizza la stringa model per istanziare il modello che deve essere interrogato e usa la domanda per le funzionalità di Retrieval-Augmented Generation, ottenendo le informazioni rilevanti dal database vettoriale chiamando findEmbedding(Embedding embedding, int maxResults, double minScore) del EmbeddingRepository, componendo il prompt per il LLM e interrogandolo. Restituisce la risposta generata dal modello.

- public List<String> getDocuments():
  metodo che riceve il contenuto del database vettoriale sotto forma di stringa chiamando getDocuments() dell'EmbeddingRepository, lo processa per estrarre il nome dei documenti presenti e li restituisce al Controller sotto forma di lista di stringhe.

- public String addDocument(MultipartFile newDocument):
  metodo che riceve processa il nuovo documento estraendone il testo, dividendolo a paragrafi e generando gli embedding di ognuno di essi. Questi dati saranno poi utilizzati per aggiornare il database vettoriale chiamando addDocument(List<Embedding> embeddings, List<TextSegment> segments) dell'EmbeddingRepository.

- public String deleteDocument(String toDelete):
  metodo che riceve il nome del documento da eliminare dal database vettoriale. Ottenuto l'attuale contenuto della base di dati, elimina le informazioni relative al documento da rimuovere, e sovrascrive il database chiamando String deleteDocument(String toDelete) dell'EmbeddingRepository, a cui passa il database risultante sotto forma di stringa.

AuthService

Descrizione: È la classe che contiene la business logic per le operazioni di autenticazione dell'utente.
Riceve le chiamate dal Controller, utilizza la classe AuthRepository per ricevere le credenziali dell'amministratore, e restituisce il risultato delle operazioni al Controller stesso.

Attributi

- private final AuthRepository authRepository.

Metodi

- public boolean checkCredential(String password):
  metodo che riceve dal Controller la password inserita dall'utente e la compara con la password presente in database, recuperata attraverso findByUsername(String username) dell'AuthRepository.

EmbeddingRepository

Descrizione: È la classe che contiene la persistence logic per operare sui dati contenuti nel database vettoriale.

Attributi

- private InMemoryEmbeddingStore<TextSegment> embeddingStore.

Metodi

- public List<EmbeddingMatch<TextSegment>> findEmbedding(Embedding embedding, int maxResults, double minScore):
  metodo che riceve la domanda embedizzata, il massimo numero di informazioni rilevanti che deve recuperare dal database vettoriale e il valore minimo di similarità semantica che queste informazioni devono avere. Recupera le informazioni dal database vettoriale in-memory, e restituisce al RagService una lista di coppie embedding rilevanti-testo relativo.

- public String getDocuments():
  metodo che accede al database vettoriale e lo restituisce al RagService convertito a stringa.

- public String addDocument(List<Embedding> embeddings, List<TextSegment> segments):
  metodo che riceve dal RagService una lista di nuovi embedding e relativi testi, inserendo le informazioni ricevute al database vettoriale.

- public String deleteDocument(String toDelete):
  metodo che riceve dal RagService il nuovo contenuto del database vettoriale, senza il documento rimosso, e lo persiste.

AuthRepository

Descrizione: È la classe che contiene la persistence logic per operare sui dati relativi alla funzionalità di autenticazione.
Implementa l'interfaccia JpaRepository, propria del modulo Spring JPA del framework Spring, e utilizza la classe Credential per avere accesso semplificato al database.

Metodi

- Credential findByUsername(String username):
  recupera le credenziali dell'admin sotto forma di oggetto della classe Credential. Questo oggetto, dal quale si può ottenere la password per l'autenticazione attraverso il suo metodo getPassword(), viene passato al AuthService, che confronterà la password in database con quella digitata dall'utente.

Basi di dati

Nella realizzazione di questo prototipo, sono stati utilizzati due diversi database: uno adibito a contenere le credenziali di accesso dell'amministratore, l'altro per memorizzare gli embedding dei documenti insieme al testo relativo.
Per entrambi sono stati utilizzati delle basi di dati in-memory persistite su disco: questo perché, dato lo stato prototipale del prodotto, permettevano di eseguire le funzionalità desiderate in modo più semplice e veloce di altri database.

H2 Database

H2 è un database management system relazionale scritto in Java con tecnologia in-memory, che permette la memorizzazione e il recupero veloce e sicuro di dati. Grazie alla sua integrazione con il modulo JPA di Spring, permette la gestione automatica e facilitata di database in applicazioni Spring Boot.
Oltre ad essere sicuro in quanto database criptato, e quindi compatibile con il ruolo di contenere dati per l'autenticazione dell'applicativo, la scelta è di questo database è data dalla veloce integrazione nel progetto con le JpaRepository, utilizzata per l'AuthRepository.

InMemoryEmbeddingStore

InMemoryEmbeddingStore è l'embedding database completamente in-memory utilizzabile con LangChain4J. In esso sono contenuti dei testi, ottenuti dal parsing e divisione in paragrafi di documenti, in coppia con gli embedding da essi generati, formati da un Embedding Model. Per ogni coppia testo-embedding, sono memorizzati dei metadati che danno informazioni aggiuntive sul contenuto del database, tra cui il nome del documento da cui provengono.
Per persistere il database, il suo contenuto è convertito in JSON e salvato su disco. Ogni volta che viene riavviato il prototipo, viene instanziato l'InMemoryEmbeddingStore a partire dal file JSON conservato.
L'utilizzo di un database vettoriale in-memory non è solo più semplice da implementare, ma è anche indicato per l'uso in prototipi. Questo perché con un quantitativo ridotto di dati memorizzati (sotto il centinaio di documenti), utilizzare un database in-memory fa sì che le performance dell'Embedding Model, nel ricercare e recuperare le informazioni rilevanti, non siano in alcun modo influenzate. Infatti, le prestazioni degli Embedding Model sono, seppur minimamente, diverse se operano in un database piuttosto che un altro. Usare un vectore store in-memory, quando la mole di dati in questione è ridotta, è quindi consigliato.

Autenticazione

Per accedere all'area relativa all'amministratore, dove poter visualizzare, aggiungere e rimuovere i documenti a cui il chatbot ha accesso, è necessario autenticarsi.
Data la sola presenza di una login (il solo admin, la funzionalità di chat è aperta ad ogni altro utente generico senza registrazione), nella schermata relativa è richiesta l'immissione della sola password.

La password dell'amministratore è stata memorizzata nella base di dati in-memory H2, implementando varie misure di sicurezza. Infatti, oltre ad essere già di per sè un database sicuro in quanto criptato, la password è stata criptata prima di essere memorizzata. Per la criptazione è stata utilizzata la funzione di hashing BCrypt, implementata grazie al modulo Security del framework Spring.

L'autenticazione utilizza il metodo HTTP POST, così da non mostrare mai in chiaro la password inserita dall'utente. Grazie alla libreria messa a disposizione da Spring Security, una volta recuperata la password inserita dall'utente e quella criptata presente nel database, esse sono comparate senza la necessità di decriptare la password salvata.

Pattern del backend

Controller-Service-Repository Pattern

Attraverso la distinzione del backend nelle classi Controller, Service e Repository, è stato possibile dividere il codice in base al suo behavior e scopo. Così facendo, è stato possibile dividere la logica di business, contenuta nelle classi Service, da quella di persistenza, propria dei Repository.

Dependency Injection

Tramite l'utilizzo del decoratore @Autowired del framework Spring, è stato applicato l'Inversion of Controll nell'applicativo. La DI consente di separare la creazione degli oggetti dalle loro dipendenze, fornendo un modo più flessibile e modulare per organizzare il codice.
Seguire questo pattern architetturale permette il decoupling del prototipo, così da rendere il codice più modulare e, potenzialmente, facile da testare. Inoltre, poiché la DI promuove la separazione dei componenti e delle loro dipendenze, il codice diventa più modulare e meno incline agli effetti collaterali. Questo semplifica la manutenzione e l'evoluzione del codice nel tempo.

Builder

Il pattern Builder è un pattern di progettazione che viene utilizzato per costruire oggetti complessi passo dopo passo. Il Builder è particolarmente utile quando si tratta di creare oggetti complessi con numerosi attributi o opzioni di configurazione. Suddividendo la costruzione dell'oggetto in passaggi separati, rende più semplice gestire la complessità e mantenere il codice pulito e comprensibile. Inoltre, questo pattern offre un'interfaccia flessibile per la creazione di oggetti, consentendo di specificare solo le opzioni desiderate e di lasciare le altre opzioni con valori di default. Questo offre una maggiore flessibilità nella creazione degli oggetti e permette di adattarli facilmente alle esigenze specifiche dell'applicazione.
L'impiego di questo pattern è stato utilizzato per l'instanziazione delle classi relative ai modelli linguistici, potendo scegliere quali valori configurare, come nome del modello o temperatura, tralasciando quelli non di interesse, come il time out.

Frontend finale

Home page

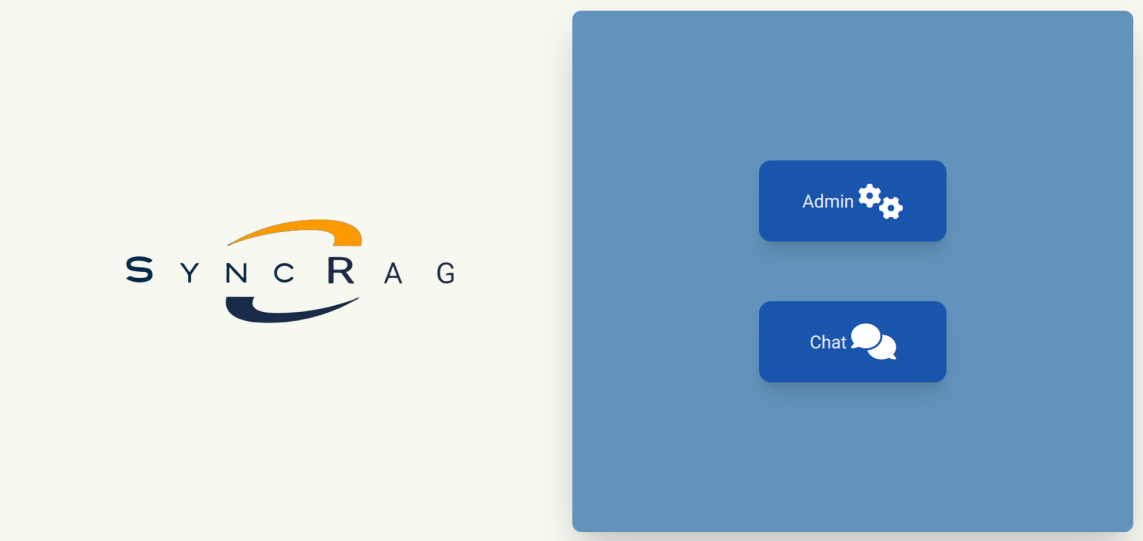

Figura 11: Interfaccia grafica finale della home page

Pagina di autenticazione

Figura 12: Interfaccia grafica finale della pagina di autenticazione

Pagina dell'amministratore

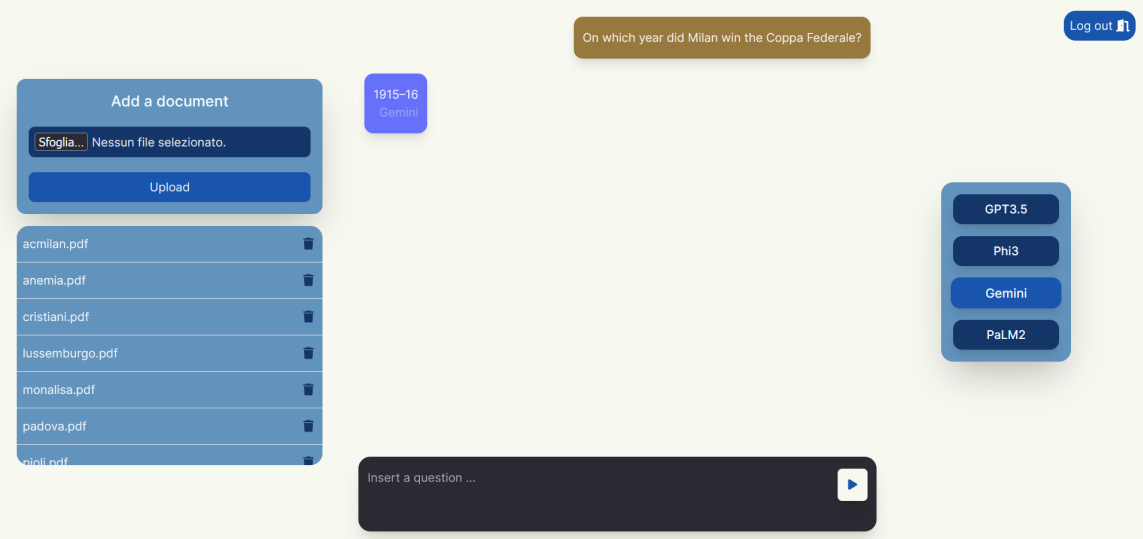

Figura 13: Interfaccia grafica finale della pagina dell'amministratore

Pagina di chat

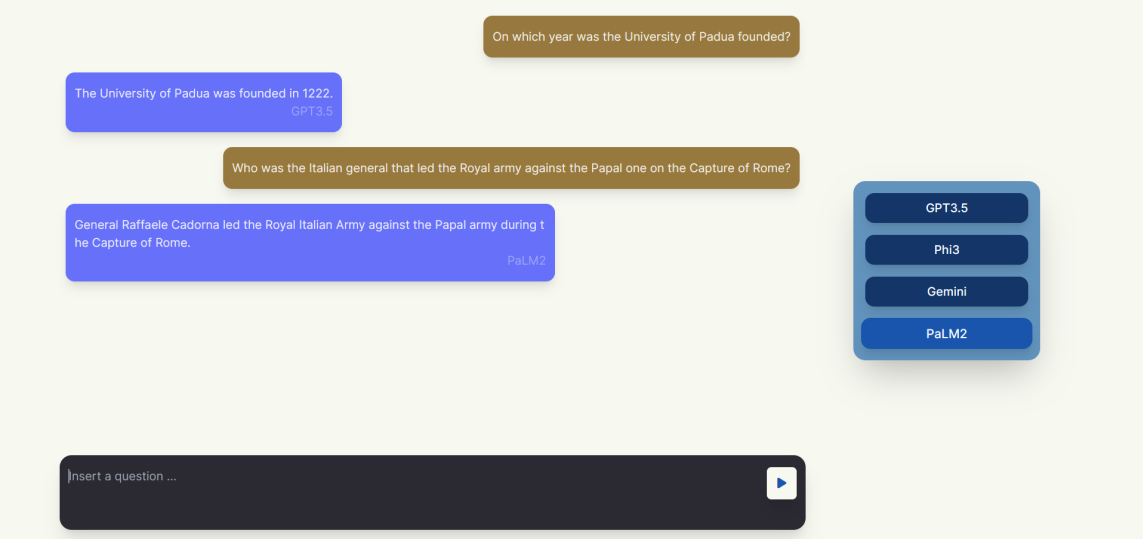

Figura 14: Interfaccia grafica finale della pagina di chat

Pattern del frontend

Componund Components

Specifico di React e di applicazioni components-based, il Compound Components è un pattern che prevede l'elevata modularizzazione del codice front-end, andando a definire singoli componenti, con un unico scopo, da inserire all'interno di componenti contenitori. Con questo approccio, viene creata un'alberatura delle componenti che prevede una relazione di padre-figlio fra le stesse.
Seguendo questo pattern, è possibile creare componenti riutilizzabili, che aumentano la flessibilità e la manutenibilità codice, evitando di codificare l'intera interfaccia grafica in un unico grande componente.

Un esempio dell'utilizzo di questo pattern è visibile nelle pagine di chat e dell'amministratore, dove tutti gli elementi in comune sono stati codificati una sola volta, e poi importati e riutilizzati nelle due pagine.

Higher-Order Components

Il High Order Components (HOC) è un pattern utilizzato tipicamente in React per incapsulare la logica di un componente, che prenderà il nome di Wrapped Comopnent, in un altro. Nel contesto di un controllo di autenticazione per una pagina, un HOC può essere utilizzato per proteggere l'accesso a una determinata pagina in base allo stato di autenticazione dell'utente.
In particolare, la pagina con accesso protetto è inglobata internamente ad un altro componente, nel quale è implementata la logica di sicurezza. Se l'autenticazione è stata verificata correttamente dal componente di alto ordine, il componente incapsulato viene correttamente renderizzato. Al contrario, l'accesso sarà bloccato, e non sarà visibile all'utente non autenticato il contenuto protetto.

Utilizzo

Per poter utilizzare il prototipo, è prima necessario svolgere le seguenti azioni:

1. recarsi nella directory frontend;
2. eseguire da terminale il comando `npm install`, per installare le dipendenze necessarie per l'esecuzione del frontend, e `npm run dev` per eseguire l'applicazione;
3. recarsi nella directory backend;
4. eseguire da terminale il comando `mvn spring-boot:run`, che eseguirà l'applicazione che espone i servizi necessari al prototipo sulla porta *8080*;
5. usare un browser per aprire *http://localhost:3000*, per accedere finalmente al prototipo.

Al primo utilizzo del prototipo in una macchina/server, sarà prima necessario impostare la password per accedere all'area dell'amministratore. Per fare questo:

1. recarsi nella directory backend;
2. eseguire da terminale il comando `mvn spring-boot:run`, che eseguirà l'applicazione che espone i servizi necessari al prototipo sulla porta *8080*;
3. usare un browser per aprire *http://localhost:8080/h2-console*, per accedere all'interfaccia del database H2;
4. accedere con username `syncrag` e password `password`;
5. eseguire i comandi SQL `'CREATE TABLE CREDENTIALS ( username VARCHAR(255) PRIMARY KEY, password VARCHAR(255) NOT NULL );'` e `'INSERT INTO CREDENTIALS (username, password) VALUES ('admin', $PASSWORD);'`, dove al posto di $PASSWORD va inserita la password desiderata, criptata con funzione di hashing Bcrypt.

Altre informazioni

5.1 Prompt

Nell'interrogare i vari modelli, è sempre utilizzato lo stesso template di prompt:

```
Answer to the following question, based ONLY on the context i'll give you.

Question:---
{{question}}
---

Context:---
{{information}}
---

ONLY IF you have no useful information you MUST answer with 'I can't provide
any answer.'.
Don't use general knowledge to give information outside the context.
```

Al posto di {{question}} viene inserita la domanda che l'utente ha inserito nel chatbot. Le informazioni rilevanti recuperate dal database vettoriale sono invece riportate nella posizione delimitata da {{information}}.

5.2 Informazioni rilevanti

Nello stabilire quali informazioni dare al modello come contesto utile per generare la risposta, è stata utilizzata la semantic similarity tra l'embedding della domanda con quello dei chunk presenti nel vectore store. Nella prima fase di selezione, vengono presi in considerazione solo le informazioni che presentano una similarità del coseno pari o superiore al 40%, così da escludere in partenza le informazioni certamente inutili. Successivamente, sono selezionati i primi 5 chunk con somiglianza maggiore, i quali vengono recuperati e inseriti nel prompt da dare al modello.

5.3 Parametri per l'instanziazione dei modelli

Phi-3 mini

```
ChatLanguageModel model = OllamaChatModel.builder()
                .baseUrl("http://localhost:11434")
                .modelName("phi3")
                .temperature(0.6)
                .timeout(Duration.ofMinutes(1))
                .build();
```

GPT 3.5 Turbo

```
ChatLanguageModel model = OpenAiChatModel.builder()
                .apiKey("demo")
                .modelName(GPT_3_5_TURBO)
                .temperature(0.6)
                .build();
```

Gemini 1.5 Pro

```
ChatLanguageModel model = VertexAiGeminiChatModel.builder()
                .project(PROJECT-NAME)
                .location("us-central1")
                .modelName("gemini-1.5-pro-preview-0514")
                .temperature(0.6F)
                .build();
```

PaLM 2 Bison

```
ChatLanguageModel model = VertexAiChatModel.builder()
                .project(PROJECT_NAME)
                .location("us-central1")
                .modelName("chat-bison-32k")
                .publisher("google")
                .endpoint("us-central1-aiplatform.googleapis.com:443")
                .temperature(0.6)
                .build();
```

5.4 Database vettoriale

Un esempio di come i dati sono memorizzati nel database vettoriale, convertito in formato JSON:


```
{"entries":[{"id":"684af8f3-2748-416e-b820-4ccb162dbb4c","embedding":{"vector":[-0.10028294, 0.02156058,
-0.089896426, 0.045987308, -0.08517339, 0.039354544, -0.038126566, 0.024594737, -0.0024825684, 0.06814546,
0.07786715, -0.03821498, 0.058066733, 0.06631604, -0.027114546, -0.01672334, -0.051882643, -0.05845935,
0.03005478, 0.014067019, -0.021481484, -0.10625454, -0.026393844, -0.07704766, 0.031230597, 0.05503411,
0.014892961, 0.011255466, -0.017610537, 0.03724367, -0.042718578, 0.045750313, 0.07864776, 0.06310859,
-0.0101201655, -0.0071547567, 0.08220105, 0.017995514, 0.037220784, 0.074548215, -0.006999618, -0.015482095,
0.062755086, 0.013979072, 0.09193837, 0.03490314, 0.08876505, 0.026195465, -0.06034913, 0.095428005, 0.008014704,
0.045964636, 0.10078499, -0.060068853, -0.030683799, 0.022353334, -0.026065782, 0.048524555, -0.015553534,
-0.0024572825, 0.07486399, -0.0366679, -0.08132293, 0.017215665, -0.030232703, -0.05085155, -0.023987817,
-0.046750274, 0.053890735, 0.0037904494, 0.066871956, -0.09057614, -0.0024872902, -0.001332385, 0.005353505,
0.11869505, 0.05070818, -0.025790853, -0.009677063, -0.062251862, 0.011300198, -0.027564442, -0.025719238,
0.027461868, -0.03142551, 0.017376216, 0.05089721, 0.050133165, 0.0027643042, 0.0133843515, -0.0032161279,
0.053730242, -0.027698796, -0.024793565, 0.018927671, -0.01648891, -0.069883406, 0.06827755, 0.0070034983,
0.010022728, 0.017837124, 0.050233714, 0.0015174358, 0.0447307, -0.017675629, -0.014466751, 0.014545249,
0.04476749, -0.018817892, 0.03880881, -0.07886634, -0.023484753, -0.10166826, -0.010596607, -0.023960125,
0.061548058, -0.019435538, -0.0032578304, 0.03153377, -0.047084793, -0.03005156, 0.02581496, -0.022161087,
0.013088655, 0.017074768, 0.05417119, -0.037161667, -9.809389E-34, -0.09287385, 0.021192132, 0.014397297,
-0.013219615, -0.023483638, -0.018732257, 0.011808089, 0.02375296, -0.015500356, -0.0102819735, -0.005045593,
3.518264E-4, 0.082637414, -0.1623152, -0.014055691, -0.058858458, -0.011350709, -0.07388533, -0.0067638904,
-0.05612241, 0.05614653, -0.022141451, 0.06605671, -0.116716035, 0.040190272, -0.016439995, -0.05257548,
-0.014566255, -0.035699055, -0.01796168, 0.07654706, -0.0042825593, -0.03714374, 0.0115005085, 0.014685103,
0.03811123, 7.373108E-4, -0.053024095, -0.01004299, -0.07467095, 0.038164012, -0.012801256, -0.02157135,
```

-0.06296981, 0.02819813, -0.023362461, -0.17475595, -0.030275749, 0.04516143, -0.078598216, 0.03556739,
0.056484584, 0.04688554, -0.007624995, 0.017060734, 0.009201439, -0.12759155, -0.009610951, 0.045232628,
-0.08906936, 0.0622969, -0.016993647, -0.085431315, 0.024996426, -0.058453295, 0.025399068, 0.08956928,
0.018015983, 0.013313997, -0.03945806, 0.11967149, -0.07457374, -0.038335346, -0.043361187, -0.02271841,
0.070843466, 0.008228385, 0.040932547, -0.02617108, 0.021407012, -0.02621667, 0.036677036, 0.06607781,
0.034870982, 0.0046323626, 0.032410238, 0.016362235, 0.066054605, -0.022204498, -0.012334222, 0.0048204563,
0.023607071, -0.0027500251, 0.026202377, 0.04054849, -1.206946E-33, 0.057378124, -0.0047503132, -0.026388798,
-0.06874393, -0.0060354844, -0.017529609, -0.07065707, -0.018931016, 0.0010069887, 0.049889628, 0.082537964,
0.07750371, -0.010307118, -0.07833603, -0.0475618, -0.0022050154, 0.018430403, -0.009880736, -0.018047033,
-0.0042034113, -0.03957734, -0.00654916, -0.022844018, -0.03020144, -0.03547145, -0.05705241, 0.09459159,
0.031267703, -0.074854195, 0.012508206, -0.064368114, -0.017576866, -0.0017534053, -0.035036564, 0.037699852,
0.0066408715, -0.08008018, 0.006436367, 0.034094706, 0.045150645, -0.03289874, 0.050171345, -0.10023248,
0.0156121105, 0.14825824, -0.02582686, 0.025223944, -0.02429635, -0.043391895, 0.00304912, 0.05082433,
0.0017433519, 0.072562605, -0.0975687, 0.121792026, 0.07953375, -0.047714233, 0.008520193, -0.006650233,
-0.08517704, -0.0050429767, -0.012028273, -0.099404305, 0.049187277, -0.030925266, 0.024468603, -0.058353942,
0.14868751, -0.09295751, -0.057653356, -0.037202056, -0.011795123, -0.07336864, 0.100086704, -0.11369111,
-0.0429888, 0.024362344, 0.030896528, 0.0069983206, 0.0018598125, -0.051065166, 0.0055742566, -0.019617008,
0.10399376, 0.049236137, -0.015130345, -0.002386048, -0.04345193, 0.01632174, 0.08585416, 0.040932473,
-0.05079669, 0.035414144, 0.01125926, 0.0053816107, -3.658272E-8, 0.020583447, 0.077843994, 0.038081758,
0.07390528, 0.08637727, 0.004316184, -0.028772492, -0.06779503, 0.13153656, 0.0020353282, -0.048579473,
0.014510406, 0.0054121166, 0.011715985, -0.049989056, 0.025342137, -0.056945704, 0.023866754, 0.0301191,
0.03553886, -0.049251877, 0.002952557, -0.014377362, 0.030448444, 0.0123793855, -0.04031022, -0.052894834,
-0.079072036, -0.060403205, -0.030302564, -0.004360896, 0.03660778, 0.070213184, -0.06588985, -1.9539882E-
4, 0.023564246, 0.095518485, -0.09231646, -0.0368691, -0.075189754, -0.017240528, 0.05580782, 0.014347192,
-0.0144126555, 0.039792556, 0.07056231, -0.034236774, -0.04325971, -0.067207545, -0.03552506, -0.011916566,
0.13032499, 0.051905595, -0.072263695, 0.047859315, -0.017831387, -0.07771918, 0.05529743, -5.45952E-5,
0.029090932, -0.021574393, 0.0076737935, -0.0029619269, -0.022498043]},"embedded":{"text":"AC Milan was
founded as Milan Foot-Ball and Cricket Club in 1899 by English expatriate Herbert Kilpin. The club claims
16 \r\nDecember of that year as their foundation date, but historical evidence seems to suggest that the
club was actually founded a \r\nfew days earlier, most likely on 13 December. However, with the club0027s
charter being lost, the exact date remains open to \r\ndebate.","metadata":{"metadata":{"absolute_directory_path":
"C:\\Path\\assoluto\\del\\documento\\embeddizato","file_name":"acmilan.pdf","index":"0"}}}}]}

È possibile notare che ogni record del database vettoriale contiene sia il chunk (frazione del documento), sia il suo embedding. Il vettore multidimensionale è utilizzato nel calcolo della semantic similarity, il testo è invece recuperato e inserito nel database.
Quando deve essere eliminato un documento, viene utilizzato il metadato ”file_name” per identificare quali dati rimuovere dal database.

5.5 Inserimento documenti

Quando viene inserito un documento nel prototipo, esso subisce un processo di elaborazione per inserire il suo contenuto nel database vettoriale.
Innanzitutto, il documento viene parserato, andando ad estrarre il testo che lo compone. Il testo è successivamente diviso in frazioni più piccole, dette chunck, che possono essere a lunghezza costante (numero di caratteri o token) o variabile (per frase, paragrafo o pagina del documento).
Ottenuti i vari chunk, essi sono processati dall'embedding model per generare i vettori embedding corrispondenti, che saranno utilizzati nella fase di ricerca delle informazioni rilevanti.
Infine, ogni chunk viene inserito in database con il relativo embedding, andando ad aggiungere dei metadati utili ad ogni record, come il nome del documento da cui proviene l'informazione.

La motivazione per cui è necessario dividere in chunk i documenti, e non embeddizarli tutti interi, è duplice. Da un lato, le dimensioni del vettore di embedding dipende dal modello che lo genera, ed è indipendente dalla grandezza del chunk: più è ampio il testo da rappresentare come vettore numerico, più complicato sarà per il vettore rappresentare le caratteristiche semantiche delle frasi. Inoltre, avere un documento intero significa fornire al modello l'intero documento come contesto utile, richiedendo che il LLM abbia una context window e delle performance maggiori rispetto alla divisione in chunk più piccoli.