Code7Crusaders

Software Development Team

Specifica Tecnica

Membri del Team:
Enrico Cotti Cottini, Gabriele Di Pietro, Tommaso Diviesti
Francesco Lapenna, Matthew Pan, Eddy Pinarello, Filippo Rizzolo

Ver	Data	Redattore	Verificatore	Descrizione
0.4	18/03/2025	Matthew Pan		Stesura sezione 3.2 e 3.3
0.3	12/03/2025	Francesco Lapenna		Prima stesura sezione 3.1
0.2	5/03/2025	Eddy Pinarello	Francesco Lapenna	Stesura sezioni 2 e 4
0.1	1/03/2025	Eddy Pinarello	Francesco Lapenna	Prima stesura del documento

Indice

1 Introduzione **4**
1.1 Scopo specifica tecnica . . . 4
1.2 Scopo del prodotto . . . 4
1.3 Glossario . . . 4
1.4 Riferimenti . . . 4
1.4.1 Riferimenti normativi . . . 4
1.4.2 Riferimenti informativi . . . 4

2 Lista tecnologie **6**
2.1 Docker . . . 6
2.2 Linguaggi di programmazione e formato dati . . . 6
2.3 Librerie . . . 6
2.4 Servizi . . . 7
2.4.1 OpenAI API . . . 7

3 Architettura **8**
3.1 Introduzione all'architettura . . . 8
3.1.1 Scopo e obbiettivi . . . 8
3.2 Architettura del sistema . . . 8
3.2.1 Architettura monolitica . . . 8
3.2.2 Architettura a microservizi . . . 9
3.2.3 Scelta del monolite esagonale . . . 9
3.3 Architettura esagonale . . . 10
3.3.1 Principi fondamentali . . . 10
3.3.2 Struttura dell'architettura esagonale . . . 10
3.3.3 Vantaggi dell'architettura esagonale . . . 10
3.3.4 Conclusione . . . 11
3.4 Moduli . . . 11
3.5 Tecnologie . . . 11
3.5.1 OpenAI API . . . 11

4 Tracciamento dei requisiti **12**
4.1 Tracciamento requisiti funzionali . . . 12
4.2 Tracciamento requisiti di vincolo . . . 15
4.3 Tracciamento requisiti di qualità . . . 16
4.4 Soddisfazione totale dei requisiti . . . 16

Elenco delle tabelle

1 Linguaggi e formati utilizzati . . . 6
2 Librerie utilizzate . . . 7
3 Tabella Requisiti funzionali soddisfatti . . . 15
4 Tabella Requisiti di vincolo soddisfatti . . . 15
5 Tabella Requisiti di qualità soddisfatti . . . 16

Elenco delle figure

1 Schema dell'architettura esagonale 10

1 Introduzione

1.1 Scopo specifica tecnica

Questo documento è rivolto a tutti gli stakeholder coinvolti nel progetto Code7Crusaders, un chatbot B2B pensato per semplificare la ricerca di prodotti all'interno dei cataloghi dei distributori. Il documento fornisce una visione dettagliata dell'architettura del sistema, dei design pattern utilizzati, delle tecnologie adottate e delle scelte progettuali effettuate. Inoltre, include diagrammi UML delle classi e delle attività per descrivere il funzionamento del sistema in modo chiaro e strutturato.

1.2 Scopo del prodotto

Lo scopo del prodotto è realizzare un **Assistente Virtuale basato su LLM** , per supportare aziende produttrici di bevande nel fornire informazioni dettagliate e personalizzate sui loro prodotti. Il sistema si rivolge principalmente ai proprietari di locali, consentendo loro di ottenere risposte rapide e precise su caratteristiche, disponibilità e dettagli delle bevande, come se interagissero con uno specialista umano.

1.3 Glossario

Per garantire una chiara comprensione della terminologia utilizzata nel documento, è stato predisposto un *Glossario*[G] in un file dedicato. Questo strumento serve a evitare ambiguità nella definizione dei termini impiegati nell'attività progettuale, offrendo descrizioni precise e condivise.

1.4 Riferimenti

1.4.1 Riferimenti normativi

- **Capitolato C7 LLM**: ASSISTENTE VIRTUALE
  `https://www.math.unipd.it/~tullio/IS-1/2024/Progetto/C7.pdf`
- **Regolamento del progetto didattico**
  `https://www.math.unipd.it/~tullio/IS-1/2024/Dispense/PD1.pdf`
- **Norme di Progetto** v.1.0
  `https://code7crusaders.github.io/docs/RTB/documentazione_interna/norme_di_progetto.html`

1.4.2 Riferimenti informativi

- **Slide Corso Ingegneria del software: Analisi dei Requisiti**
  `https://www.math.unipd.it/~tullio/IS-1/2024/Dispense/T05.pdf`
- **Slide Corso Ingegneria del software: Diagrammi delle classi**
  `https://www.math.unipd.it/~rcardin/swea/2023/Diagrammi%20delle%20Classi.pdf`
- **Slide Corso Ingegneria del software: Diagrammi dei casi d'uso**
  `https://www.math.unipd.it/~rcardin/swea/2022/Diagrammi%20Use%20Case.pdf`
- **Glossario**[G] v.1.0
  `https://code7crusaders.github.io/docs/RTB/documentazione_interna/glossario.html`
- **Analisi LLM**
  `https://code7crusaders.github.io/docs/altri_documenti/analisi_modelli_firmato.html`

- **Analisi framework frontend**
  `https://code7crusaders.github.io/docs/altri_documenti/analisi_frontend_firmato.html`
- **Analisi framework backend**
  `https://code7crusaders.github.io/docs/altri_documenti/analisi_framework_backend.html`
- **Analisi database Vettoriale**
  `https://code7crusaders.github.io/docs/altri_documenti/analisi_dbvettoriale.html`
- **LangChain**[G]
  `https://python.langchain.com/docs/introduction/`
- **OpenAI**
  `https://openai.com/`

2 Lista tecnologie

Questa sezione ha lo scopo di offrire una panoramica delle tecnologie adottate per la realizzazione del sistema software. Vengono analizzati in dettaglio le piattaforme, gli strumenti, i linguaggi di programmazione, i framework e altre risorse tecnologiche utilizzate nel corso dello sviluppo.

2.1 Docker

È una piattaforma di containerizzazione leggera che facilita lo sviluppo, il testing e il rilascio delle applicazioni, fornendo un ambiente isolato e riproducibile. Viene utilizzato per creare ambienti di sviluppo uniformi, migliorare la scalabilità delle applicazioni e semplificare la gestione delle risorse.

2.2 Linguaggi di programmazione e formato dati

Nome	Versione	Descrizione	Impiego
Python	3.0	Linguaggio di programmazione ad alto livello, dinamico e interpretato	Sviluppo backend, gestione API ed embedding model
JavaScript	ES6	Linguaggio di programmazione interpretato, principalmente utilizzato per lo sviluppo frontend	Sviluppo frontend, interattività delle pagine web, utilizzo di React
SQL	-	Linguaggio di programmazione per la gestione e manipolazione di database relazionali	Gestione database, query, manipolazione dati
YAML	1.2	Formato di serializzazione dati leggibile dall'uomo	Configurazione, script GitHub Actions
JSON	-	Formato di interscambio dati leggero e leggibile dall'uomo	Gestione database, scambio dati tra client e server

Tabella 1: Linguaggi e formati utilizzati

2.3 Librerie

Python		
Nome	**Versione**	**Impiego**
`Flask`	3.1.0	Framework per applicazioni web in Python.
`Flask-Cors`	5.0.0	Estensione per Flask per gestire le richieste CORS.
`langchain-core`	0.3.31	Modulo per la gestione dei documenti in LangChain.
`langchain-openai`	0.3.1	Integrazione di OpenAI con LangChain.
`requests`	2.32.3	Libreria per effettuare richieste HTTP.
`python-dotenv`	1.0.1	Gestione delle variabili d'ambiente da file `.env`.
`faiss-cpu`	1.9.0.post1	Gestione dei database vettoriali FAISS in LangChain.
`numpy`	2.2.2	Libreria per il calcolo scientifico e la manipolazione di array.

Nome	Versione	Impiego
`openai`	1.60.0	Libreria per interfacciarsi con l'API di OpenAI.
`SQLAlchemy`	2.0.37	Toolkit SQL per Python.
JavaScript		
-	-	-

Tabella 2: Librerie utilizzate

2.4 Servizi

2.4.1 OpenAI API

L'API di OpenAI fornisce accesso a modelli di intelligenza artificiale avanzati, tra cui modelli di embedding. Un embedding model è un tipo di modello di machine learning che trasforma dati di input, come parole o frasi, in vettori di numeri in uno spazio continuo a bassa dimensione. Questi vettori catturano le caratteristiche semantiche dei dati di input, permettendo di misurare la similarità tra diversi input in modo efficiente.

Vantaggi: L'utilizzo di embedding models offre numerosi vantaggi, tra cui:

- **Efficienza**: I vettori di embedding permettono di rappresentare dati complessi in modo compatto e computazionalmente efficiente.
- **Versatilità**: Possono essere utilizzati in una vasta gamma di applicazioni, tra cui il processamento del linguaggio naturale (NLP), la raccomandazione di contenuti e la classificazione dei dati.

Casi d'uso: Gli embedding models sono utilizzati in vari casi d'uso, tra cui:

- **Ricerca di documenti**: Migliorano la ricerca di documenti trovando risultati più rilevanti basati sulla similarità semantica.
- **Raccomandazione di contenuti**: Personalizzano le raccomandazioni di contenuti in base alle preferenze dell'utente.
- **Classificazione del testo**: Aiutano nella classificazione automatica di testi in categorie predefinite.

Impiego del progetto: Nel progetto, l'API di OpenAI viene utilizzata per convertire il testo in token e generando embedding che rappresentano le caratteristiche semantiche del testo in uno spazio vettoriale.

3 Architettura

3.1 Introduzione all'architettura

3.1.1 Scopo e obbiettivi

La presente sezione ha lo scopo di fornire una visione d'insieme dell'architettura del sistema, evidenziandone i principi guida e le scelte progettuali che ne hanno determinato la struttura. In particolare, si intende:

- Definire il contesto in cui opera il sistema, evidenziando i requisiti funzionali e non funzionali che hanno condotto alla scelta di una specifica architettura.
- Orientare i lettori (sviluppatori, progettisti e stakeholder) verso una comprensione chiara delle componenti principali e delle interazioni che caratterizzano il sistema.
- Porre le basi per la discussione delle scelte di design, evidenziando come queste possano rispondere alle esigenze di scalabilità, sicurezza, manutenibilità e performance.
- Descrivere le motivazioni alla base delle scelte tecnologiche e dei modelli architetturali adottati.

3.2 Architettura del sistema

La progettazione dell'architettura del sistema ha richiesto un'approfondita analisi delle due principali opzioni architetturali disponibili: **monolitica** e **a microservizi**. La scelta dell'architettura è stata guidata da una serie di fattori, tra cui la natura dell'applicazione, il volume di traffico previsto, i costi di sviluppo e manutenzione e la necessità di scalabilità. In questa sezione verranno analizzati nel dettaglio i pro e i contro di entrambe le soluzioni, per poi motivare la decisione finale.

3.2.1 Architettura monolitica

Un'architettura monolitica si basa su un'unica codebase che incorpora tutti i componenti dell'applicazione, tra cui l'interfaccia utente, la logica di business e il livello di accesso ai dati. Questo approccio, tradizionalmente adottato nello sviluppo software, è particolarmente indicato per applicazioni di piccola e media complessità, in cui i costi di separazione dei componenti in unità indipendenti non sono giustificati.

Vantaggi

- **Semplicità di sviluppo e gestione**: Un'unica codebase permette di mantenere una visione centralizzata del sistema, semplificando lo sviluppo, il testing e il debugging.
- **Minori costi di infrastruttura**: Non è necessario investire in strumenti di orchestrazione, load balancing o gestione della comunicazione tra microservizi.
- **Deployment più semplice**: L'intero sistema viene distribuito come un'unica unità, evitando problemi di coordinamento.
- **Prestazioni migliori per bassi volumi di traffico**: L'assenza di chiamate di rete tra microservizi riduce la latenza.

Svantaggi

- **Scalabilità limitata**: Non è possibile scalare singole componenti separatamente.
- **Maggiore impatto degli errori**: Un bug in una parte dell'applicazione può compromettere l'intero sistema.
- **Difficoltà nell'adozione di nuove tecnologie**: L'aggiornamento di singole parti è complesso poiché l'intero stack è integrato.

3.2.2 Architettura a microservizi

L'architettura a microservizi suddivide il sistema in componenti indipendenti, ognuno responsabile di una funzionalità specifica. Ogni microservizio comunica con gli altri attraverso API, permettendo un alto grado di indipendenza e flessibilità nello sviluppo.

Vantaggi

- **Scalabilità orizzontale**: Ogni microservizio può essere scalato indipendentemente.
- **Flessibilità nello sviluppo**: Permette l'adozione di tecnologie diverse per ciascun servizio.
- **Maggiore resilienza**: Un errore in un microservizio non compromette l'intero sistema.
- **Facilità di manutenzione**: È possibile distribuire aggiornamenti senza dover ripubblicare l'intera applicazione.

Svantaggi

- **Maggiore complessità gestionale**: L'orchestrazione dei microservizi richiede strumenti avanzati.
- **Comunicazione tra servizi**: Introduce latenza e potenziali colli di bottiglia.
- **Deployment più complesso**: Coordinare il rilascio di più servizi è più oneroso.
- **Costi di sviluppo più elevati**: La frammentazione del sistema richiede maggiore sforzo di progettazione e testing.

3.2.3 Scelta del monolite esagonale

Dopo un'analisi approfondita, il team di sviluppo ha deciso di adottare un'**architettura monolitica esagonale**. Questa scelta è stata motivata dai seguenti fattori:

1. **Basso carico di utenti**: L'applicazione è destinata a un contesto B2B con un numero limitato di utenti concorrenti.
2. **Semplicità di gestione**: La manutenzione di un monolite è più diretta rispetto a un sistema distribuito.
3. **Riduzione dei costi operativi**: L'assenza di strumenti di orchestrazione riduce significativamente i costi di infrastruttura.
4. **Velocità di sviluppo**: Un'unica codebase consente iterazioni rapide senza dipendenze tra servizi separati.
5. **Evoluzione graduale verso microservizi**: Adottando un'architettura **esagonale**, il sistema può essere trasformato gradualmente in microservizi senza riscrivere tutto.

3.3 Architettura esagonale

L'**architettura esagonale**, nota anche come *Ports and Adapters*, è un pattern architetturale l'obiettivo di rendere il software più flessibile, testabile e indipendente dalle tecnologie esterne. Questo approccio enfatizza la separazione tra la logica di business e le interfacce di comunicazione con il mondo esterno.

3.3.1 Principi fondamentali

L'architettura esagonale si basa su tre concetti chiave:

- **Isolamento della logica di business**: Il core dell'applicazione è indipendente dai dettagli implementativi esterni.
- **Utilizzo di porte e adattatori**: Le *porte* definiscono le interfacce per la comunicazione tra il core e il mondo esterno, mentre gli *adattatori* implementano queste interfacce per specifiche tecnologie.
- **Sostituibilità delle dipendenze**: È possibile cambiare database, framework web o altre dipendenze senza impattare il core.

3.3.2 Struttura dell'architettura esagonale

L'architettura esagonale può essere rappresentata con tre livelli principali:

1. **Core (Dominio e Logica di Business)**: Contiene le regole fondamentali dell'applicazione.
2. **Porte (Ports)**: Interfacce che definiscono i punti di ingresso e uscita del sistema.
3. **Adattatori (Adapters)**: Implementazioni concrete delle porte per database, servizi esterni e UI.

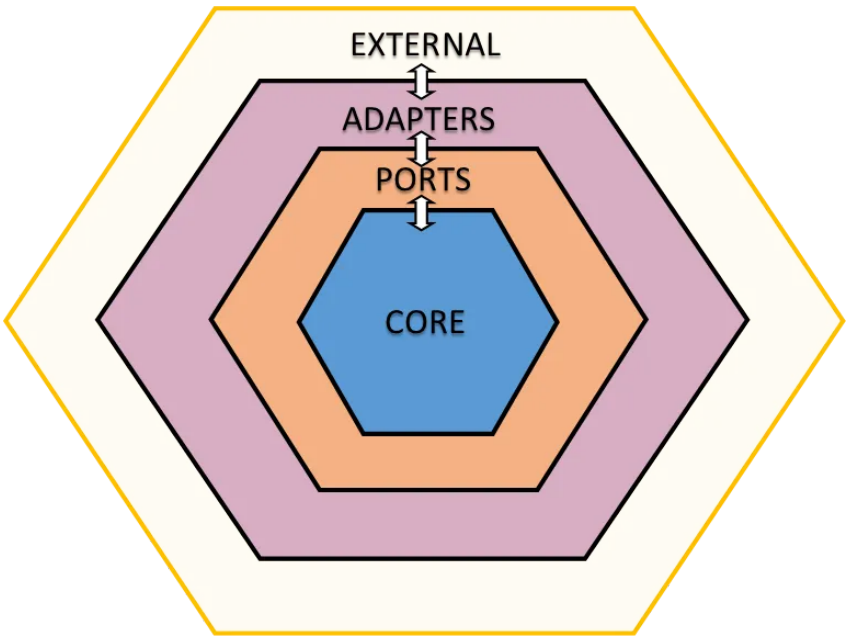

Figura 1: Schema dell'architettura esagonale

3.3.3 Vantaggi dell'architettura esagonale

Adottare un'architettura esagonale comporta diversi benefici:

- **Maggiore manutenibilità**: Il codice è modulare e separato.

- **Facilità di test**: Il core dell'applicazione può essere testato isolatamente.
- **Indipendenza dalle tecnologie**: Cambiare framework o database ha un impatto minimo.
- **Flessibilità evolutiva**: Permette di trasformare gradualmente il monolite in microservizi.

3.3.4 Conclusione

L'architettura esagonale garantisce modularità e sostenibilità del sistema nel lungo termine, permettendo di scalare senza impattare la stabilità complessiva dell'applicazione.

3.4 Moduli

3.5 Tecnologie

3.5.1 OpenAI API

4 Tracciamento dei requisiti

In questa sezione vengono descritti i requisiti del sistema e il loro tracciamento. Ogni requisito è identificato da un codice univoco che ne facilita la gestione e il monitoraggio. I requisiti sono suddivisi in categorie in base alla loro natura (funzionali, di qualità, di vincolo) e alla loro importanza (obbligatori, desiderabili, facoltativi). Di seguito viene presentata una tabella che traccia i requisiti funzionali del sistema, indicando per ciascuno di essi il codice identificativo, la descrizione e lo stato di soddisfacimento.
I requisiti sono codificati come segue: **R[Tipo][Importanza][Numero]**
Dove **Tipo** può essere:

- **F (funzionale)**
- **Q (di qualità)**
- **V (di vincolo)**

Importanza può essere:

- **O (obbligatorio)**
- **D (desiderabile)**
- **F (facoltativo)**

Numero è un numero identificativo univoco del requisito.

4.1 Tracciamento requisiti funzionali

Codice	Descrizione	Stato
RFO1	L'amministratore inserisce dalla pagina di gestione i dati semantici aziendali da cui apprendere la conoscenza da file in formato .pdf.	-
RFO2	L'amministratore inserisce dalla pagina di gestione i dati semantici aziendali da cui apprendere la conoscenza da file in formato .txt.	-
RFO3	I testi recuperati dai documenti verranno suddivisi in blocchi, ovvero pezzi più piccoli di dati che rappresentano una piccola porzione del contesto.	-
RFO4	I vettori generati verranno memorizzati all'interno di un database vettoriale e opportunamente indicizzati.	-
RFO5	Da un'interfaccia utente della web app, viene catturata una domanda da parte dell'utente.	-
RFO6	La domanda viene inoltrata al sistema attraverso delle API REST risiedenti in un Web Server.	-
RFO7	La rappresentazione vettoriale viene utilizzata per effettuare una ricerca all'interno del database vettoriale da dove vengono reperiti i vettori più simili.	-
RFO8	La domanda viene inviata al sistema LLM tramite API.	-
RFO9	Viene attesa la risposta dall'LLM tramite API.	-
RFO10	Attraverso API REST, il sistema inoltra la risposta all'account dell'utente.	-
RFO11	L'utente deve essere in grado di ottenere informazioni riguardo un prodotto attraverso la conversazione con il bot.	-

Codice	Descrizione	Stato
RFO12	L'utente deve essere in grado di ottenere informazioni riguardo una serie di prodotti attraverso la conversazione con il bot.	-
RFO13	La conversazione tra utente e bot deve essere salvata.	-
RFO14	L'utente deve essere in grado di visualizzare una delle conversazioni precedentemente salvate.	-
RFO15	L'utente deve essere in grado di riprendere una delle conversazioni precedentemente salvata.	-
RFO16	L'utente o l'amministratore devono poter accedere al sistema inserendo Username e Password.	-
RFO17	L'utente si registra inserendo Username e Password.	-
RFO18	Gli input del form di registrazione devono essere sanificati per prevenire attacchi SQL Injection.	-
RFO19	Gli input del form di accesso devono essere sanificati per prevenire attacchi SQL Injection.	-
RFO20	L'utente deve essere in grado di dare un feedback (thumbsup/thumbsdown) sulla qualità della conversazione dopo averla provata.	-
RFO21	L'accesso alla dashboard dei "template di domanda e risposta" è consentito solo agli utenti con ruolo di amministratore.	-
RFO22	Dopo l'accesso da parte dell'amministratore, la pagina di gestione mostra la dashboard dei "template di domanda e risposta".	-
RFO23	Un "template di domanda e risposta" è formato da una domanda (possibilmente una domanda posta frequentemente che l'amministratore decide di inserire per risparmiare una chiamata al modello) associata ad una corrispondente risposta.	-
RFO24	L'amministratore deve essere in grado di creare un template, che è formato da una domanda associata ad una corrispondente risposta.	-
RFO25	L'amministratore deve essere in grado di modificare uno dei template esistenti.	-
RFO26	L'amministratore deve essere in grado di eliminare un template esistente.	-
RFO27	Il sistema deve poter fermare la creazione di un template invalido, ovvero quando il template non rispetta il formato Json.	-
RFF28	L'amministratore deve poter accedere alla dashboard di monitoraggio delle metriche.	-
RFF29	L'accesso alla dashboard delle metriche delle run è consentito solo agli utenti con ruolo di amministratore.	-
RFF30	Dopo l'accesso da parte dell'amministratore, la pagina di gestione mostra la dashboard delle metriche delle run.	-
RFF31	L'amministratore deve poter selezionare criteri di filtro per visualizzare solo le run di interesse.	-
RFF32	Il sistema deve permettere la selezione di filtri come ID, nome, input, data di inizio e fine, errore, output, tag, numero di token, costo.	-

Codice	Descrizione	Stato
RFF33	Una volta selezionati i filtri, il sistema deve aggiornare la visualizzazione senza ricaricare l'intera pagina.	-
RFF34	Se nessun filtro è selezionato, il sistema mostra le prime dieci run per impostazione predefinita.	-
RFF35	Dopo aver applicato i filtri, l'amministratore deve poter visualizzare le metriche principali delle run selezionate.	-
RFF36	Il sistema deve mostrare le metriche principali delle run filtrate (ID, nome, input, data di inizio e fine, errore, output, tag, token totali, costo totale).	-
RFF37	La visualizzazione deve essere chiara e strutturata, con possibilità di ordinare le colonne.	-
RFO38	L'amministratore deve poter visualizzare i feedback dati dagli utenti.	-
RFO39	Il sistema deve poter rifiutare l'importazione dati di file non compatibili, ovvero file non nel formato pdf o txt.	-
RFO40	L'utente deve poter eliminare una conversazione precedentemente effettuata.	-
RFO41	L'utente deve poter mandare richieste di assistenza per poter parlare con un operatore umano.	-
RFO42	L'accesso alla dashboard delle richieste di assistenza è consentito solo agli utenti con ruolo di amministratore.	-
RFO43	Dopo l'accesso da parte dell'amministratore, la pagina di gestione mostra la dashboard delle richieste di assistenza.	-
RFO44	L'amministratore deve poter visualizzare le richieste di assistenza ricevute da parte dell'utente.	-
RFO45	L'amministratore deve poter segnalare ad altri amministratori che una richiesta è stata presa in carico.	-
RFD46	L'amministratore deve essere in grado di poter rispondere all'utente tramite contatto via e-mail.	-
RFF47	Le metriche delle run del chatbot devono essere esportabili in JSON.	-
RFF48	Le metriche della run devono includere ID univoco della run, nome assegnato alla sessione, dati di input elaborati dal modello, timestamp di avvio e completamento dell'esecuzione, eventuali errori incontrati, risultato generato dal modello, numero totale di token utilizzati e stima dei costi basata sul consumo di token.	-
RFO49	Il bot per rispondere a una domanda deve ricordarsi i messaggi precedenti nella singola conversazione.	-
RFD50	Il sistema deve notificare l'utente quando la memoria per le chat salvate è piena e non è possibile salvare ulteriori conversazioni.	-
RFO51	L'utente seleziona una delle domande tra quelle predefinite.	-
RFO52	L'utente deve essere in grado di visualizzare una lista delle conversazioni precedentemente salvate.	-
RFO53	La lunghezza massima dell'username è di 256 caratteri.	-
RFO54	La lunghezza massima della password è di 256 caratteri.	-

Codice	Descrizione	Stato
RFO55	Il Sistema rifiuta la registrazione di un nuovo account con username già presente.	-

Tabella 3: Tabella Requisiti funzionali soddisfatti

4.2 Tracciamento requisiti di vincolo

Codice	Descrizione	Stato
RVO1	Il chatbot deve rispondere con il contesto dato dai file di allenamento (pdf o file di testo inseriti)	-
RVO2	LLM deve essere integrato tramite API	-
RVO3	LLM utilizzato deve essere quello di OpenAI	-
RVO4	Deve essere usato un database relazionale	-
RVO5	Deve essere gestito il salvataggio delle chat precedenti con tutti i messaggi in esse tramite un database relazionale con PostgreSQL	-
RVO6	Deve essere implementato un database vettoriale	-
RVO7	Deve essere implementato un database vettoriale FAISS per poter rendere possibile la ricerca con contesto dall'LLM	-
RVO8	Deve essere implementato un embedding model	-
RVO9	L'embedding model deve essere quello di OpenAI	-
RVO10	Deve essere implementata una WebApp che permetta di comunicare con il chatbot	-
RVO11	L'interfaccia deve essere costruita utilizzando componenti funzionali React	-
RVO12	Si deve creare un backend che gestisca le chiamate HTTP, il database vettoriale e il database relazionale con Flask	-
RVO13	La gestione dello stato locale deve essere implementata tramite useState	-
RVO14	La WebApp deve utilizzare React Router per gestire la navigazione tra le pagine	-
RVO15	Gli stili devono essere gestiti tramite CSS inline o con className per garantire modularità	-
RVO16	La comunicazione tra componenti deve essere gestita inviando funzioni come props	-
RVO17	La WebApp deve essere responsiva e adattarsi dinamicamente alle dimensioni della finestra	-
RVO18	La gestione dei blocchi di testo vettorializzati deve essere gestita tramite Faiss	-
RVD19	Le metriche delle run del chatbot devono essere recuperate tramite Langsmith	-
RVO20	Bisogna usare la libreria LangChain per la interazione con i modelli LLM e Embedding	-

Tabella 4: Tabella Requisiti di vincolo soddisfatti

4.3 Tracciamento requisiti di qualità

Codice	Descrizione	Stato
RQO1	Schema di progettazione della base di dati	-
RQO2	Codice prodotto in formato sorgente reso disponibile tramite repository pubblici	-
RQO3	Documentazione riassuntiva delle metriche e dei risultati	-
RQO4	Il software deve essere testato con una copertura di codice minima dell'80% e una copertura dei rami dell'80%, con un obiettivo ottimale del 100%	-
RQO5	Il 90% dei test deve essere superato come requisito minimo, mentre l'obiettivo ottimale è il 100%	-
RQO6	La metodologia di sviluppo deve seguire il paradigma del Test Driven Development (TDD), garantendo che il codice venga scritto partendo dai test	-

Tabella 5: Tabella Requisiti di qualità soddisfatti

4.4 Soddisfazione totale dei requisiti

Il gruppo Code7Crusaders ha soddisfatto - su -, arrivando ad una copertura del -%.

Soddisfatti	Non soddisfatti
-	-

Grafico 1: Requisiti soddisfatti rispetto al totale.

Per quanto riguarda la copertura dei requisiti obbligatori, la copertura rilevata è di - su - requisiti, arrivando quindi ad un -% sul totale.

Soddisfatti	Non soddisfatti
-	-

Grafico 2: Requisiti obbligatori soddisfatti rispetto al totale.

In termini di soddisfacimento dei requisiti desiderabili, è stata raggiunta una copertura del -%, con - su -.

Soddisfatti	Non soddisfatti
-	-

Grafico 3: Requisiti desiderabili soddisfatti rispetto al totale.

Per quanto concerne l'adempimento dei requisiti opzionali, abbiamo conseguito una percentuale del -% sul totale, con - su - requisiti considerati.

Soddisfatti	Non soddisfatti
-	-

Grafico 4: Requisiti opzionali soddisfatti rispetto al totale.