Università degli Studi di Padova
Laurea: Informatica
Corso: Ingegneria del Software
Anno Accademico: 2024/2025

Gruppo: SWEg Labs
Email: gruppo.sweg@gmail.com

Specifica Tecnica

Versione 1.0.0

Stato	Approvato
Redazione	Federica Bolognini Michael Fantinato Giacomo Loat Filippo Righetto Riccardo Stefani Davide Verzotto
Verifica	Federica Bolognini Michael Fantinato Giacomo Loat Filippo Righetto Riccardo Stefani Davide Verzotto
Proprietario	Giacomo Loat
Uso	Esterno
Destinatari	Prof. Tullio Vardanega Prof. Riccardo Cardin *AzzurroDigitale Srl*

Registro delle modifiche

Versione	Data	Descrizione	Autore	Verificatore
1.0.0	...	Modifiche a ...	...	Giacomo Loat
...	...	Modifiche a ...	...	...
0.1.10	05-03-25	Inizializzazione della struttura delle sezioni §**3.3**, §**3.4** e §**3.5**	Riccardo Stefani	Giacomo Loat
0.1.9	25-02-25	Aggiunta sezione §**3.2** riguardante l'architettura di deployment	Filippo Righetto	Riccardo Stefani
0.1.8	24-02-25	Aggiunta sezione §**3** e stesura architettura logica	Davide Verzotto	Riccardo Stefani
0.1.7	13-02-25	Aggiunto PostgreSQL alla sezione §**2.1.3**	Giacomo Loat	Riccardo Stefani
0.1.6	11-02-25	Sistemati i link presenti nella sezione §**1.5** seguendo i consigli del professor Vardanega	Riccardo Stefani	Giacomo Loat
0.1.5	10-02-25	Aggiunti PyTest e Jasmine alla sezione §**2.2.2**	Riccardo Stefani	Giacomo Loat
0.1.4	10-02-25	Aggiunto FastAPI alla sezione §**2.1.1**, Angular e Node Docker alla sezione §**2.1.2** e GPT-4o alla sezione §**2.1.4**	Michael Fantinato	Riccardo Stefani
0.1.3	10-02-25	Aggiunti Python e LangChain alla sezione §**2.1.1** e aggiunto Docker alla sezione §**2.1.4**	Giacomo Loat	Riccardo Stefani
0.1.2	08-02-25	Scrittura sezione §**1**	Federica Bolognini	Riccardo Stefani
0.1.1	06-02-25	Scrittura sezioni §**2.1.3** e §**2.2.1**	Riccardo Stefani	Giacomo Loat
0.1.0	06-02-25	Creazione del documento	Riccardo Stefani	Federica Bolognini

Tabella 1: Registro delle modifiche

Indice

Elenco delle figure

Elenco delle tabelle

1 Introduzione

1.1 Scopo del documento

Questo documento fornisce linee guida per gli sviluppatori incaricati dell'estensione o della manutenzione del prodotto.
Al suo interno sono raccolte tutte le informazioni sui linguaggi e le tecnologie adottate, sull'architettura del sistema e sulle scelte progettuali effettuate.

1.2 Scopo del prodotto

Nell'ultimo anno vi è stato un cambiamento repentino nello sviluppo e nell'applicazione dell'*Intelligenza Artificiale*$_G$ all'elaborazione e raccomandazione dei contenuti alla generazione di essi, come immagini, testi e tracce audio. Il *capitolato*$_G$ C9, "BuddyBot", pone come obiettivo la realizzazione di un applicativo che permetta di porre interrogazioni in linguaggio naturale sullo stato attuale dei progetti software in lavorazione, ricevendo una risposta il quanto più precisa. Tale risposta dovrà essere generata tramite un *LLM*$_G$ collegato. Tale software sarà fruibile attraverso un'*applicazione web*$_G$, dove l'utente potrà interrogare il chatbot sullo stato attuale del codice e della documentazione dei progetti software nelle piattaforme utilizzate per il loro sviluppo.

1.3 Glossario

Al fine di prevenire ed evitare possibili ambiguità nei termini e acronimi presenti all'interno della documentazione, è stato realizzato un glossario nel file *glossario_v2.0.0.pdf* in grado di dare una definizione precisa per ogni vocabolo potenzialmente ambiguo. All'interno di ogni documento i termini specifici, che quindi hanno una definizione all'interno del *Glossario*$_G$, saranno contrassegnati con una $_G$ aggiunta a pedice e trascritti in corsivo. Tale prassi sarà rispettata solamente per la prima occorrenza del termine in una determinata sezione del documento.

1.4 Miglioramenti al documento

La revisione e l'evoluzione del documento sono aspetti fondamentali per garantirne la *qualità*$_G$ e l'adeguatezza nel tempo. Questo consente di apportare modifiche in base alle esigenze concordate tra i membri del gruppo e il *proponente*$_G$. Di conseguenza, questa versione del documento non può essere considerata definitiva o completa, in quanto soggetta a possibili aggiornamenti futuri.

1.5 Riferimenti

1.5.1 Riferimenti normativi

- **Norme di Progetto v.2.0.0**;
- **Capitolato d'appalto C9 - BuddyBot (slide 3-18)**:
  https://www.math.unipd.it/ tullio/IS-1/2024/Progetto/C9.pdf *(Ultimo accesso: 03/04/2025)*;
- **Regolamento progetto didattico (slide 2-25)**:
  https://www.math.unipd.it/ tullio/IS-1/2024/Dispense/PD1.pdf *(Ultimo accesso: 03/04/2025)*;

1.5.2 Riferimenti informativi

- **Glossario v.2.0.0**;

2 Tecnologie coinvolte

Questa sezione fornisce un'analisi esaustiva delle tecnologie impiegate nel progetto in questione, comprendendo le procedure, gli strumenti e le librerie necessarie per lo sviluppo, il testing e la distribuzione del prodotto. Saranno discusse le tecnologie utilizzate per implementare sia il backend che il frontend, la gestione dei dati e l'integrazione con i servizi previsti.

2.1 Tecnologie utilizzate per la codifica

2.1.1 Strumenti per il backend

Nome	Versione	Descrizione
Python	3.13.1	Linguaggio di programmazione ad alto livello, interpretato, orientato agli oggetti e multiparadigma.
LangChain	0.3.18	LangChain è un *framework*$_G$ open-source progettato per sviluppare applicazioni che sfruttano i *Large Language Models*$_G$.
FastAPI	0.115.6	FastAPI è un *framework*$_G$ web moderno e performante per lo sviluppo di *API*$_G$ in *Python*$_G$. Sfrutta le funzionalità avanzate di *Python* per la dichiarazione dei tipi, la validazione automatica dei dati e la generazione di documentazione interattiva, semplificando lo sviluppo e il mantenimento di applicazioni scalabili.

Tabella 2: Strumenti per il backend

2.1.2 Strumenti per il frontend

Nome	Versione	Descrizione
Angular	19.0.6	Angular è un *framework*$_G$ open source sviluppato da Google per la realizzazione di applicazioni web a pagina singola. Basato su *Typescript*$_G$, offre un'architettura modulare e funzionalità avanzate per la creazione di interfacce utente dinamiche e scalabili.
Node Docker	22-alpine	Il container *Docker*$_G$ utilizzato si basa sull'immagine ufficiale di *Node.js*$_G$ 22-alpine. Questa immagine, costruita su *Alpine Linux*$_G$, offre un ambiente di esecuzione leggero e performante, ideale per applicazioni in produzione, grazie alla sua ridotta impronta e all'ottimizzazione delle risorse.

Tabella 3: Strumenti per il frontend

2.1.3 Strumenti di gestione dei dati

Nome	Versione	Descrizione
Chroma	0.6.4	Chroma è un *database vettoriale*$_G$ progettato per memorizzare e gestire vettori ad alta dimensionalità. È ottimizzato per operazioni di ricerca e recupero efficienti, dunque aiuta a ridurre il numero di documenti di contesto da inviare all'*LLM*$_G$ selezionando solo quelli più rilevanti per la query corrente grazie alla ricerca di *similarità*$_G$ tra quest'ultima e i documenti salvati come vettori.
PostgreSQL	17.4	*PostgreSQL*$_G$, o Postgres, è un potente sistema di gestione di database relazionali ad oggetti open-source. È noto per la sua solidità, affidabilità e ricchezza di funzionalità. Supporta estensioni avanzate, transazioni complesse, e integrazioni con vari linguaggi di programmazione.

Tabella 4: Strumenti di gestione dei dati

2.1.4 Strumenti di integrazione e di supporto

Nome	Versione	Descrizione
GPT-4o	GPT-4	GPT-4o è un modello di linguaggio avanzato sviluppato da *OpenAI*$_G$, capace di comprendere ed elaborare testi complessi scritti in linguaggio naturale. Grazie alla sua architettura basata su deep learning, offre elevata coerenza, contestualizzazione e creatività nella generazione di contenuti.
Docker	27.3.1	Sistema di containerizzazione che permette la distribuzione di software in ambienti isolabili, dove ogni contenitore viene gestito in modo isolato rispetto agli altri.

Tabella 5: Strumenti di integrazione e di supporto

2.2 Strumenti per l'analisi del codice

2.2.1 Strumenti per l'analisi statica

2.2.2 Strumenti per l'analisi dinamica

Nome	Versione	Descrizione
Pylint	3.3.4	Pylint è un analizzatore di codice statico e permette di analizzare codice *Python*$_G$ senza eseguirlo. Controlla la presenza di errori, applica uno standard di codifica e cerca di dare suggerimenti su come il codice potrebbe essere modificato.
SonarQube for IDE	4.15.1	Si tratta di un'estensione per *IDE*$_G$ che aiuta a rilevare e correggere i problemi di qualità durante la scrittura del codice, individuando i difetti in modo perchè possano essere corretti prima del commit del codice. Non è specifico per linguaggio, bensì supporta tanti linguaggi differenti.
ESLint	9.16.0	ESLint è uno strumento di analisi del codice statico per identificare e segnalare pattern trovati nel codice *JavaScript*$_G$ e *TypeScript*$_G$, cioè uno dei linguaggi utilizzati da *Angular*$_G$. Aiuta a mantenere uno stile di codifica coerente e a prevenire errori comuni, fornendo suggerimenti per migliorare la qualità del codice.

Tabella 6: Strumenti per l'analisi statica

Nome	Versione	Descrizione
PyTest	8.3.4	PyTest è un framework di testing per *Python*$_G$ che consente di scrivere test semplici e scalabili. Supporta test unitari, funzionali e di integrazione, offrendo funzionalità avanzate come fixture, parametri e plugin per estendere le sue capacità.
Jasmine	5.4.0	Jasmine è un framework di testing per *JavaScript*$_G$ che permette di scrivere test unitari in modo semplice e leggibile. Fornisce un'ampia gamma di funzionalità per la scrittura di test asincroni, la gestione delle aspettative e la creazione di suite di test modulari, aiutando a garantire la qualità e la robustezza del codice. È particolarmente utile per testare applicazioni sviluppate con *Angular*$_G$, grazie alla sua integrazione con il framework.

Tabella 7: Strumenti per l'analisi dinamica

3 Architettura

Questa sezione fornisce una descrizione dettagliata dell'architettura del prodotto software, illustrando le scelte progettuali adottate per garantire la corretta realizzazione del sistema. Saranno presentati le principali scelte e *pattern architetturali*$_G$, e le *componenti software*$_G$ che compongono il sistema.

3.1 Architettura logica

L'architettura adottata nella realizzazione dell'applicativo si basa sul modello di architettura esagonale. Il livello di *business*$_G$ del software è quindi indipendente dagli altri componenti, ovvero nulla presente all'esterno della logica di business può conoscere la sua *implementazione*$_G$. Questo principio alla base dell'architettura esagonale ci permette di ottenere un prodotto software facilmente testabile e *manutenibile*$_G$. Le componenti che devono restare indipendenti sono rappresentate dal *Domain business model*$_G$, che non comunica mai direttamente con l'esterno. Gli elementi che permettono il funzionamento dell'architettura esagonale sono i seguenti:

- ***Controller***$_G$: contiene l'*application logic*$_G$ del sistema: gestisce le richieste in ingresso, valida i dati di tipo *DTO*$_G$ ricevuti in input, e li adatta verso un tipo di dato di business. Il *Controller* ha poi il compito di chiamare uno *UseCase* per eseguire la logica di business, adattare l'output in un oggetto DTO e restituirlo al client.
- ***UseCase***$_G$: rappresenta un caso d'uso specifico del sistema, che viene implementato da un *Service* per realizzare la logica di business.
- ***Service***$_G$: contiene la *business logic*textsubscript***G*** del sistema: esegue operazioni specifiche esclusive del dominio di interesse e delega le interazioni con le altre componenti ai *Port*. I *Service* possono interagire unicamente con tipi di dato di business, garantendo l'indipendenza della logica di business dal resto del sistema.
- ***Port***$_G$: definisce le interfacce attraverso le quali i *Service* interagiscono con il mondo esterno, permettendo il salvataggio e recupero di dati persistenti senza modificare la logica di business.
- ***Adapter***$_G$: implementa una o più interfacce definite dai *Port*, permettendo la comunicazione tra la logica di business e le tecnologie esterne di persistent logic. L'*adapter* si occupa di adattare i dati ricevuti dalla logica di business in un tipo *Entity*$_G$ adatto per la persistenza, e viceversa.
- ***Repository***$_G$: contiene la *persistent logic*textsubscript***G*** del sistema: gestisce la persistenza dei dati, interagendo con un database o altre forme di storage per salvare e recuperare le informazioni necessarie. I tipi di dato gestiti dal *repository* sono gli Entity, che rappresentano i dati persistenti del sistema.

È stato scelto di utilizzare un'architettura esagonale per i seguenti motivi:

- **Facilità di test**: l'architettura esagonale permette di testare facilmente la business logic, in quanto è possibile sostituire gli adapter con degli *stub*$_G$ o dei *mock*$_G$;
- ***Manutenibilità***$_G$: l'architettura esagonale permette di mantenere la business logic indipendente dagli altri componenti, facilitando la manutenzione del codice;
- ***Scalabilità***$_G$: l'architettura esagonale permette di aggiungere nuovi adapter senza dover modificare la business logic.

3.2 Architettura di Deployment

Per determinare l'architettura di deployment più adatta all'applicativo, si è tenuto conto del contesto reale in cui verrà utilizzato. Poiché il sistema è destinato all'utilizzo in un azienda, dove non si prevedono significative espansioni o modifiche strutturali dopo l'installazione, la scelta di un'architettura monolitica è risultata la più appropriata. Questa soluzione, oltre a essere perfettamente in linea con le esigenze del prodotto, semplifica le fasi di progettazione, sviluppo e test. Inoltre, rispetto a un'architettura a microservizi, il monolite evita complessità che sarebbero state difficili da gestire, anche considerando le competenze attuali del team di sviluppo.

Il deployment del prodotto viene gestito attraverso la containerizzazione con ***Docker Compose***$_G$. Questa scelta consente di semplificare l'installazione dell'applicativo, fornendo un ambiente preconfigurato in cui tutte le dipendenze sono già risolte. In questo modo, vengono predisposti automaticamente tutti i servizi necessari per garantire il corretto funzionamento del sistema, riducendo le difficoltà legate alla configurazione manuale.

3.3 Architettura di dettaglio

3.3.1 Architettura della generazione di una risposta

...

3.3.2 Architettura dell'aggiornamento automatico del database vettoriale

...

3.3.3 Architettura dell'inizializzazione, refresh e scroll di Angular

...

3.3.4 Architettura dell'aggiornamento del rendering grafico durante la chat

...

3.3.5 Architettura frontend dell'aggiornamento del badge di segnalazione dell'esito dell'ultimo aggiornamento automatico

...

3.3.6 Architettura backend dell'aggiornamento del badge di segnalazione dell'esito dell'ultimo aggiornamento automatico

...

3.3.7 Architettura del salvataggio dei messaggi nello storico

...

3.3.8 Architettura del recupero dei messaggi dallo storico

...

3.3.9 Architettura della generazione di domande per proseguire la conversazione

...

3.4 Design pattern utilizzati

3.4.1 Architettura esagonale

...

3.4.2 Facade

...

3.4.3 Dependency Injection

...

3.4.4 MVVM

...

3.5 Descrizione delle classi

3.5.1 Backend

3.5.1.1 Controller

...

3.5.1.2 Use Case

...

3.5.1.3 Service

...

3.5.1.4 Port

...

3.5.1.5 Adapter

...

3.5.1.6 Repository

...

3.5.2 Frontend

3.5.2.1 Component

...

3.5.2.2 Service

...