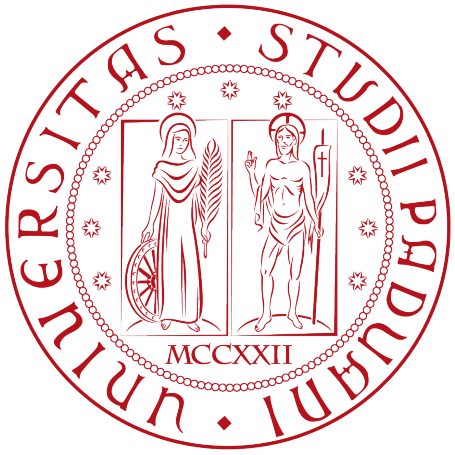
Università degli Studi di Padova

Dipartimento di Matematica "Tullio Levi-Civita"

Corso di Laurea in Informatica



Una piattaforma basata su IA Generativa per la creazione di documenti tecnici

Tesi di laurea

Relatore
Prof. Tullio Vardanega

Laureando
Alberto Michelazzo
Matricola 2010007

Anno Accademico 2023-2024

Sommario

Il presente documento descrive il lavoro svolto durante il periodo di *stage*, della durata di circa trecentoventi ore, dal laureando Alberto Michelazzo presso l'azienda Zero12 s.r.l, nel periodo compreso tra il 25 settembre ed il 22 novembre 2024.
L'obiettivo principale del progetto era quello di andare a realizzare una piattaforma *Web*, che tramite l'utilizzo di sistemi di intelligenza artificiale Generativa, fosse in grado di produrre in automatico dei documenti tecnici contenenti la descrizione delle attività di sviluppo di servizi e infrastrutture *cloud*.
Il progetto è stato sviluppato seguendo le metodologie *Agile* e *Scrum*, tutte le componenti implementate sono state opportunamente documentate e il loro corretto funzionamento è stato testato.

Struttura del documento

La seguente relazione è strutturata nei seguenti capitoli:

Capitolo 1, Contesto aziendale: Il capitolo descrive i prodotti e servizi offerti dall'azienda, la propensione aziendale all'innovazione. Illustra inoltre la metodologia di lavoro utilizzata, gli strumenti e le tecnologie adottate.

Capitolo 2, Progetto di stage: Il capitolo descrive l'idea del progetto di stage, gli obiettivi ed i vincoli imposti dall'azienda, oltre alle motivazioni che hanno portato a scegliere questo progetto.

Capitolo 3, Svolgimento dello stage: Il capitolo descrive le attività svolte durante il periodo di stage ed i risultati ottenuti.

Capitolo 4, Retrospettiva finale: Il capitolo descrive gli obiettivi raggiunti e le difficoltà incontrate. Valuta inoltre le competenze acquisite dallo stagista e riporta un confronto tra università e mondo lavorativo.

Alla fine del documento si trovano le seguenti sezioni:

- **Acronimi e abbreviazioni:** Contiene i collegamenti alle definizioni dei relativi termini contenute nel **Glossario**;
- **Glossario:** Contiene le definizioni dei termini tecnici;
- **Bibliografia:** Contiene i riferimenti bibliografici utilizzati durante la stesura del documento.

Convenzioni tipografiche

All'interno di questo documento ho adottato le seguenti convenzioni tipografiche:

- Gli acronimi, le abbreviazioni e i termini ambigui o di uso non comune menzionati vengono definiti nel **Glossario**, situato alla fine del presente documento;
- Riporto ogni termine in lingua diversa dall'italiano in *corsivo*;
- Riporto ogni termine rilevante in **grassetto**.

"Le vent se lève! Il faut tenter de vivre."

— Paul Valéry

Ringraziamenti

Innanzitutto, vorrei esprimere la mia gratitudine al Prof. Tullio Vardanega, mio relatore, per l'aiuto e il sostegno fornitomi durante la stesura di questa tesi.
Senza il suo prezioso contributo, questo lavoro non sarebbe stato realizzabile.

Desidero ringraziare di cuore i miei genitori per il loro amore incondizionato, il loro sostegno costante e per essere stati sempre al mio fianco, soprattutto nei momenti più difficili durante i miei anni di studio.

Un ringraziamento speciale va a mio fratello, per essere stato una presenza costante ed un punto di riferimento insostituibile durante il nostro percorso universitario.
Condividere questa esperienza con lui ha reso ogni traguardo più gratificante e ogni sfida più facile da affrontare.

Infine, vorrei esprimere la mia gratitudine a tutti i miei amici e amiche, con cui ho condiviso anni indimenticabili, ricchi di avventure, momenti di spensieratezza e incoraggiamento nei momenti più difficili.

Padova, Dicembre 2024 Alberto Michelazzo

Indice

Elenco delle figure

Elenco delle tabelle

Capitolo 1

Contesto aziendale

1.1 Introduzione all'azienda

Zero12 S.r.l. è una *startup* italiana fondata nel 2012, con attualmente due sedi operative: una a Padova (dove ho effettuato lo *stage*) ed una ad Empoli.
Zero12 è *partner* *Amazon Web Services* (AWS) e si occupa di sviluppare soluzioni *cloud native* per i propri clienti.
Amazon Web Services è una piattaforma di servizi *cloud* di proprietà di *Amazon* che offre molti servizi tra i quali ad esmepio potenza di calcolo, archiviazione di dati, distribuzione di contenuti ed altre funzionalità per aiutare le aziende a scalare e crescere, viene ampiamente utilizzata da Zero12 per lo sviluppo dei progetti.
L'azienda si occupa di progettare e realizzare applicazioni *web* e *mobile*, fornendo servizi di consulenza e sviluppo *software* per la realizzazione di progetti *cloud* e *serverless*.
Ogni progetto viene svolto in tutte le sue parti dal *team* Zero12, dalla progettazione fino al suo *deployment* e supporto.
Poco prima del mio arrivo, Zero12 è stata acquisita da UAN Company s.r.l., un'azienda che opera nel settore dell'informatica e della consulenza aziendale, questo sta portando non pochi cambiamenti all'interno dell'organizzazione aziendale.

1.2 Servizi e prodotti offerti

Zero12 offre vari servizi e prodotti richiesti dai propri clienti, provenienti da diversi settori e con esigenze differenti.
Di seguito una panoramica di tali servizi e prodotti:

- **Sviluppo di *software custom*:** Zero12 si occupa di sviluppare software su misura per i propri clienti, partendo dall'analisi dei requisiti fino alla realizzazione ed al supporto del prodotto finale.

- **AWS *Cloud Services*:** Zero12 è partner AWS e offre servizi di consulenza e sviluppo di soluzioni *cloud native* per i propri clienti.

- ***Machine learning*:** Zero12 offre servizi di sviluppo di modelli di *machine learning* per l'analisi dei dati e l'automazione di processi, ognuno creato secondo

le esigenze del cliente.

- ***Innovation advisory*:** Zero12 offre servizi di consulenza per l'innovazione tecnologica e la trasformazione digitale delle aziende, aiutando i clienti a scegliere le tecnologie più adatte alle proprie esigenze.

1.3 *Way of Working*

1.3.1 Metodologia di lavoro

Durante il mio periodo di *stage* ho potuto sperimentare in prima persona la metodologia di lavoro adottata da **Zero12** per lo sviluppo dei progetti.
L'azienda utilizza una metodologia di sviluppo *Agile*, utilizzando il *framework Scrum*, che permette di gestire progetti complessi e di portata variabile, garantendo una maggiore flessibilità e adattabilità ai cambiamenti.
Ogni progetto viene diviso in *sprint* di durata variabile, solitamente di una o due settimane, durante le quali vengono sviluppate le funzionalità concordate con il cliente.
Durante il mio *stage* la lunghezza di ogni *sprint* è stata di una settimana, per poter avere un *feedback* più rapido sul lavoro svolto.
Ogni progetto viene gestito da un *team* composto da un *Product Owner*, uno *Scrum Master* ed uno o più *developer*, ognuno con un ruolo ben definito all'interno del progetto.

Il flusso che ogni progetto segue è il seguente:

- ***User Stories*:** il *Product Owner* definisce le funzionalità richieste dal cliente sotto forma di *user stories*.
  Le *User Stories* sono brevi descrizioni delle funzionalità richieste dal cliente, scritte in modo da essere comprensibili sia per il cliente che per il *team* di sviluppo. Queste saranno poi inserite all'interno del *Product Backlog*, ovvero la lista delle attività da svolgere all'interno del progetto.
- ***Sprint Planning*:** all'inizio di ogni *sprint* il *team* si riunisce per pianificare le attività da svolgere durante la settimana, selezionando le *User Stories* da sviluppare, in base alla loro priorità ed alla loro complessità.
- ***Daily Standup*:** ogni giorno il *team* si riunisce per fare il punto della situazione, ognuno dei membri del *team* riporta come procede il lavoro, cosa è stato fatto il giorno precedente e cosa si prevede di fare durante la giornata.
- ***Sprint Review*:** alla fine di ogni *sprint* il *team* si riunisce con il cliente per mostrare le funzionalità sviluppate durante la settimana, assicurandosi che il lavoro sia stato svolto efficientemente e che soddisfi le aspettative del cliente.
- ***Sprint Retrospective*:** alla fine di ogni *sprint* il *team* si riunisce per fare il punto della situazione dello *sprint* appena terminato, valutando il lavoro svolto e cercando miglioramenti per i prossimi *sprint*.

1.3.2 Strumenti adottati

Strumenti organizzativi

Jira

Jira è un *Issue Tracking System* (ITS) che permette la gestione delle attività di progetto in modo *Agile*, fa parte della suite di prodotti di *Atlassian*.
Un ITS permette ai membri del team di visualizzare le attività assegnate, aggiornare lo stato di avanzamento e comunicare con gli altri membri del *team* sul lavoro svolto.
Durante lo *stage* l'ho utilizzato per la creazione di *epic* e *stories*, delle loro sotto *task*, la pianificazione delle attività ed il tracciamento dei requisiti.
La Figura 1.1 mostra alcune delle *stories* che ho creato durante l'attività di pianificazione.

Type	Key	Summary	Status	Labels
	ZSGD-1	Accesso alla piattaforma	DONE	
	ZSGD-3	Registrazione al sistema	BACKLOG	Facoltativo
	ZSGD-4	Errore di accesso in caso di credenziali errate	DONE	Obbligatorio
	ZSGD-2	Login nel sistema	DONE	Obbligatorio
	ZSGD-5	Piattaforma ottimizzata per mobile	DONE	Obbligatorio
	ZSGD-47	Logout dal sistema	DONE	
	ZSGD-6	Gestione dei progetti	DONE	
	ZSGD-7	Funzione ricerca e filtraggio progetti	DONE	Desiderabile
	ZSGD-12	Rigenerazione specifici capitoli	DONE	Desiderabile
	ZSGD-11	Visualizzazione versioni precedenti progetto	DONE	Facoltativo
	ZSGD-13	Rimozioni capitoli non necessari	BACKLOG	Facoltativo
	ZSGD-48	Eliminazione progetto generato	DONE	Obbligatorio
	ZSGD-8	Visualizzazione singolo progetto	DONE	Obbligatorio
	ZSGD-9	Visualizzazione lista progetti	DONE	Obbligatorio
	ZSGD-10	Rigenerazione progetto	DONE	Obbligatorio

Figura 1.1: *Jira* - Lista delle *stories*

Confluence

Confluence è un'applicazione di collaborazione che permette la creazione di documentazione in modo collaborativo tra i componenti di un *team* di lavoro, fa parte della suite di prodotti di *Atlassian*.
Durante lo *stage* l'ho utilizzato per la creazione di documentazione relativa al progetto, come ad esempio la documentazione tecnica ed il documento di analisi progettuale.

Visual Studio Code

Visual Studio Code (da qui in poi *VSCode*) è un *Integrated Development Environment* (IDE) sviluppato da *Microsoft*, che permette la scrittura del codice in diversi linguaggi di programmazione, è stato utilizzato per andare a sviluppare tutto il codice del progetto utilizzando il linguaggio *TypeScript*.
Un IDE permette di scrivere, testare ed effettuare il *debugging* del codice in un'unica applicazione, fornendo strumenti per facilitare lo sviluppo del *software*; *VSCode* è uno degli *Integrated Development Environment* più utilizzati.
La Figura 1.2 mostra l'interfaccia di sviluppo di *VSCode*, il codice mostrato è relativo alla funzione di generazione di un progetto.

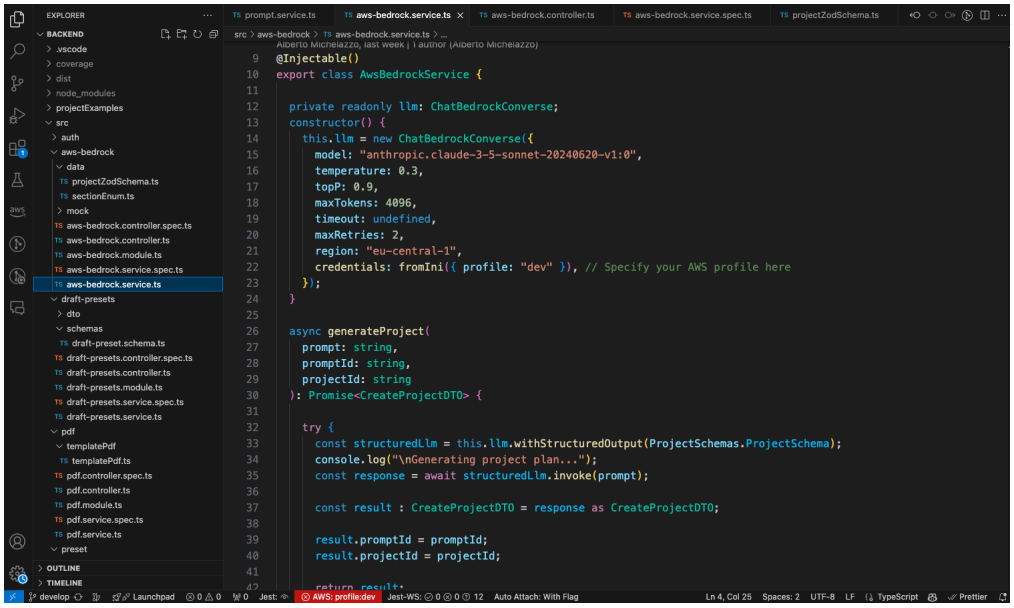


Figura 1.2: *VSCode* - Interfaccia di sviluppo

GitHub

GitHub è una piattaforma di *hosting* per progetti *software* che utilizzano il sistema di controllo di versione *Git*. Grazie a questa piattaforma è possibile andare a creare *repositories* per il proprio progetto, permettendo la collaborazione tra i membri del *team* ed il tracciamento delle modifiche effettuate.
È inoltre possibile andare a suddividere il progetto in *branch*, per andare a sviluppare le funzionalità tramite cosidetti *feature branch*.
Un *branch* è una copia del codice principale del progetto, che permette di sviluppare nuove funzionalità senza influenzare il codice principale.
Terminata la codifica di una *feature*, è possibile effettuare una *Pull Request* (PR) per andare ad unire il codice sviluppato con il *branch* principale, dopo la revisione da parte di un collega.
Una PR permette di discutere delle modifiche effettuate, garantire che il codice sia conforme alle *best practices* del progetto e che non siano presenti errori.
La Figura 1.3 mostra una delle *repositories* che ho creato per il progetto.

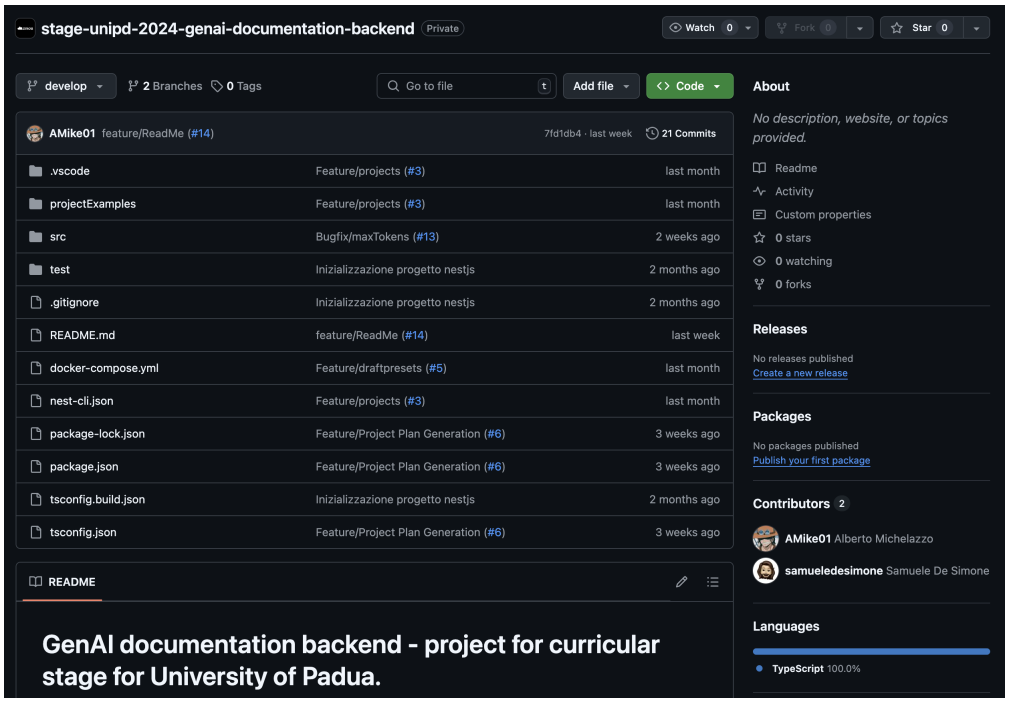


Figura 1.3: *GitHub* - Una delle *repositories* create

Strumenti di comunicazione

Slack

Slack è un'applicazione di messaggistica istantanea che permette di comunicare con i propri colleghi in ambito aziendale.
Tramite la creazione di specifici canali è possibile organizzare le conversazioni per progetto, permettendo una comunicazione più efficace tra i membri del *team*.
Come si può vedere nella Figura 1.4, nel caso del mio *stage* è stato creato un canale dedicato al mio progetto, insieme al mio tutor ed ad un'altra figura aziendale, per poter comunicare in modo diretto e veloce in caso di problematiche o dubbi durante lo sviluppo.

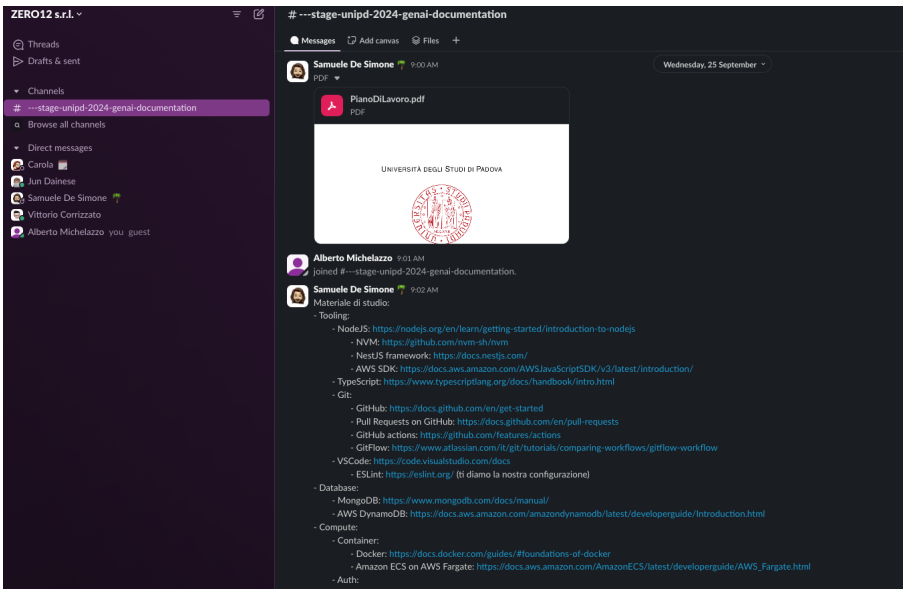


Figura 1.4: *Slack* - Canale dedicato al progetto

Microsoft Teams

Microsoft Teams è un'applicazione di collaborazione che permette la comunicazione tra i membri del *team*, la condivisione di *file* e la gestione di riunioni.
Durante il mio *stage* l'ho utilizzato per la gestione delle riunioni settimanali con il tutor aziendale, per discutere dello stato di avanzamento del progetto e per ricevere *feedback* sul lavoro svolto, quando non era possibile effettuare un incontro di persona.

1.4 Tecnologie utilizzate

1.4.1 Tecnologie di sviluppo

HTML

Il linguaggio di *markup* standard utilizzato per la creazione di pagine *web* è *HyperText Markup Language* (HTML).
Questo linguaggio consente di strutturare i contenuti di una pagina *web*, definendo la disposizione ed il formato di elementi come testo, immagini, *link*, tabelle, liste, moduli e molto altro.

Attraverso l'uso di elementi o *tag*, racchiusi tra parentesi angolari (< >), è possibile specificare la semantica e l'organizzazione del contenuto. Ad esempio, i *tag* <*p*> indicano un paragrafo, mentre i *tag* <*img*> consentono di inserire immagini.
Grazie alla sua struttura intuitiva, HTML è facile da imparare ed è utilizzato come base per la creazione di qualsiasi sito *web*.

Un aspetto fondamentale di HTML è la possibilità di collegare risorse esterne, come fogli di stile *Cascading Style Sheets* (CSS) per la gestione dell'aspetto visivo, e *file JavaScript* per l'aggiunta di funzionalità interattive.
Questo permette di separare il contenuto dalla presentazione e dalla logica, favorendo un design modulare e flessibile.

CSS

CSS è un linguaggio di stile utilizzato per descrivere l'aspetto e la formattazione dei componenti di una pagina *web*.
Consente di controllare il *layout*, i colori, i *font*, gli spazi, le dimensioni e molti altri aspetti visivi degli elementi definiti in *HyperText Markup Language*, separando il contenuto dalla presentazione.

Grazie a CSS, è possibile applicare stili uniformi a più pagine *web* collegando un unico foglio di stile esterno.
Questo approccio facilita la gestione e la modifica dell'aspetto del sito, migliorandone la coerenza visiva e riducendo il tempo necessario per apportare cambiamenti.
Versioni più recenti, come *CSS3*, hanno introdotto nuove caratteristiche avanzate, tra cui gradienti, ombre, bordi arrotondati e la possibilità di gestire contenuti multimediali e design responsivo (*responsive design*) per adattare le pagine *web* a diversi dispositivi e dimensioni dello schermo.
Queste caratteristiche fanno CSS uno strumento fondamentale nello sviluppo *web* moderno.

TypeScript

TypeScript è un *"superset"* di *JavaScript* che introduce funzionalità avanzate, come la tipizzazione statica e strumenti per migliorare la qualità del codice.
Grazie ad un compilatore dedicato, il codice *TypeScript* viene tradotto in codice *JavaScript* standard, rendendolo eseguibile su qualsiasi *browser* o ambiente che supporti *JavaScript*.

Questo linguaggio è diventato un punto di riferimento per molte aziende nello sviluppo di applicazioni *web*, grazie alla sua capacità di individuare errori già durante l'attività di scrittura del codice, evitando problemi durante l'esecuzione. La tipizzazione statica consente infatti di definire chiaramente i tipi delle variabili, delle funzioni e degli oggetti, riducendo al minimo i *bug* legati ad errori di tipo.
Inoltre, strumenti come l'autocompletamento ed il controllo dei tipi durante lo sviluppo migliorano significativamente la produttività dei programmatori, offrendo un'esperienza di sviluppo più sicura e affidabile.

L'adozione di *TypeScript* è in costante crescita anche grazie alla sua integrazione con *framework* popolari come *React*, *Angular* e *Vue.js*. Questo lo rende una scelta ideale per progetti su larga scala, dove la manutenibilità del codice e la collaborazione tra *team* di sviluppo sono essenziali. Di conseguenza, molte aziende preferiscono *TypeScript* per la sua capacità di prevenire errori prima che raggiungano l'utente finale, ottimizzando così i tempi di sviluppo e riducendo i costi di manutenzione del software.

La Figura 1.5 riporta le principali funzionalità di *TypeScript* rispetto a *JavaScript*.

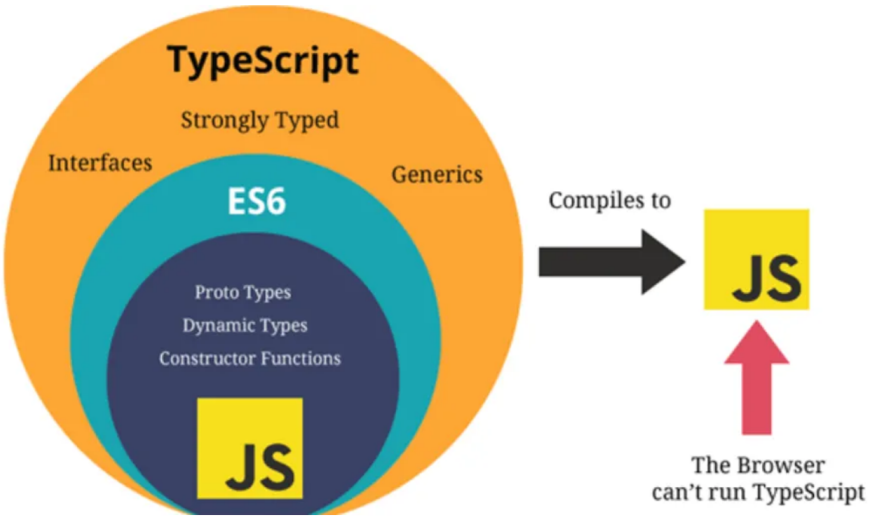


Figura 1.5: Funzionalità di *TypeScript* rispetto a *JavaScript*
JavaScript vs TypeScript. URL:
https://medium.com/geekculture/typescript-vs-javascript-e5af7ab5a331

React

Per la codifica del *Front-end* del progetto ho utilizzato *React* con *TypeScript*, per garantire una maggiore qualità e manutenibilità del codice.
Il *Front-end* di un'applicazione rappresenta la parte visibile ed interattiva con cui gli utenti interagiscono direttamente. Include l'interfaccia utente, come pulsanti, menu, testi, immagini, e la logica che gestisce le loro interazioni.
Esso è responsabile dell'esperienza utente, garantendo che l'applicazione sia funzionale, veloce e visivamente accattivante su diversi dispositivi e *browser*.

React è una libreria *JavaScript*, sviluppata da *Meta*, progettata per la creazione di interfacce utente dinamiche e reattive.
Utilizzata nello sviluppo di applicazioni *web* moderne, consente di costruire interfacce tramite un approccio basato su componenti, ovvero blocchi modulari di codice che rappresentano singole parti dell'interfaccia, come pulsanti, schede o intere sezioni di una pagina.
Questi componenti sono codificati utilizzando *file* con estensioni *JavaScript* (*JSX*) o *TypeScript* (*TSX*), permettendo di combinare logica e *markup* in modo intuitivo.

React è spesso utilizzato per creare *Single Page Application*: applicazioni *web* che caricano un'unica pagina HTML ed aggiornano dinamicamente il contenuto in base all'interazione dell'utente, senza richiedere il caricamento completo della pagina.
Questo approccio garantisce un'esperienza utente fluida e veloce, simile a quella di un'applicazione *desktop*, le *Single Page Application* infatti gestiscono la navigazione attraverso il *routing client-side*.

L'integrazione di *TypeScript* con *React* è particolarmente utile in progetti complessi. Grazie alla tipizzazione statica di *TypeScript*, è possibile scrivere componenti più robusti e manutenibili, riducendo il rischio di errori e migliorando l'autocompletamento ed il *refactoring*, garantendo che il comportamento dell'applicazione sia sempre prevedibile.

Material UI

Una libreria di componenti *React* che implementa il *Material Design*, uno stile di *design* moderno sviluppato da *Google*, progettato per garantire coerenza visiva, semplicità e accessibilità nelle interfacce utente.

Questa libreria fornisce un'ampia gamma di componenti predefiniti, come pulsanti, *cards*, tabelle, e *dialogs*, pronti all'uso e facilmente personalizzabili per adattarsi alle specifiche esigenze di un progetto.
Ogni componente è costruito per integrarsi perfettamente con l'ecosistema di *React*, sfruttando appieno l'approccio basato su componenti e le funzionalità degli *state* e dei *props*.

Utilizzare questa libreria in un progetto *React* semplifica la creazione di interfacce moderne, professionali e *responsive*, riducendo significativamente il tempo necessario allo sviluppo, garantendo un'esperienza utente coerente e di alta qualità.

Axios

Axios è una libreria *JavaScript* progettata per semplificare la comunicazione tra il *Front-end* ed il *Back-end* di un'applicazione, consentendo di effettuare richieste *Hypertext Transfer Protocol* (HTTP) come `GET`, `POST`, `PUT` e `DELETE` in modo semplice e personalizzabile.
Supporta nativamente le *Promises*, rendendo la gestione delle operazioni asincrone più chiara e leggibile, ed offre funzionalità avanzate come l'impostazione di intestazioni personalizzate, gestione automatica dei *JSON*, *timeout* configurabili e la possibilità di intercettare richieste e risposte per aggiungere logiche specifiche.

HTTP è uno *standard* di comunicazione utilizzato su *Internet* per il trasferimento di informazioni tra un *client* (come un *browser*) ed un *server*.
Ogni interazione si basa su un modello di richiesta-risposta: il *client* invia una richiesta al *server* specificando l'azione desiderata (es. recuperare dati o inviarli), ed il *server* risponde con le informazioni richieste o con un messaggio d'errore.

Il *Back-end* di un'applicazione rappresenta la parte non visibile dall'utente, responsabile della logica di elaborazione dei dati, della gestione del *database* e dell'esposizione delle API necessarie per interagire con il *Front-end*.
Utilizzando *Axios*, il *Front-end* può comunicare con il *Back-end*, inviando dati come input dell'utente o recuperando contenuti dinamici, consentendo lo sviluppo di applicazioni *web* moderne e interattive.

Le API sono interfacce che consentono la comunicazione tra diverse applicazioni o componenti software, permettendo loro di scambiarsi dati e funzionalità.
Espongono un insieme di operazioni che possono essere invocate da altre applicazioni, nascondendo la complessità del sistema sottostante.
Ad esempio, possono essere utilizzate per recuperare dati da un *database* o inviare informazioni ad un sistema di pagamento.

Le API sono fondamentali per lo sviluppo di applicazioni moderne, poiché semplificano lo scambio di dati e permettono l'integrazione di diverse funzionalità, garantendo allo stesso tempo scalabilità e interoperabilità tra sistemi.

Vite

Vite è uno strumento di sviluppo rapido e leggero che semplifica la configurazione e ottimizza il flusso di lavoro nel processo di sviluppo di applicazioni *web*.
Utilizza *ES Modules* per un caricamento veloce delle risorse ed una gestione ottimizzata dei moduli, riducendo i tempi di avvio e di aggiornamento durante lo sviluppo.

Quando utilizzato con *React*, *Vite* facilita la creazione di applicazioni configurando automaticamente il progetto, gestendo il *bundling* ed il supporto per l'*hot module replacement*, che permette di visualizzare istantaneamente le modifiche nel codice senza dover ricaricare la pagina, migliorando notevolmente l'esperienza di sviluppo.

React-PDF

React-PDF è una libreria per il *rendering* di documenti PDF direttamente all'interno di applicazioni *React*.

Questa libreria consente di visualizzare *file* PDF come componenti *React*, rendendo possibile l'integrazione di documenti interattivi nelle pagine *web* senza la necessità di *software* esterni.

Con *React-PDF*, è possibile caricare e mostrare file PDF, navigare tra le pagine, zoomare e persino aggiungere funzionalità interattive, come la gestione di eventi per rendere l'esperienza utente più dinamica.
Inoltre, la libreria offre il supporto per l'integrazione di funzionalità avanzate come la visualizzazione di PDF remoti o caricati dinamicamente.
React-PDF è ideale per applicazioni che necessitano di visualizzare *report*, documenti o qualsiasi altro tipo di contenuto PDF direttamente nel *browser*.

Styled Components

Styled Components è una libreria per *React* che consente di scrivere stili CSS direttamente all'interno dei componenti, utilizzando una sintassi che combina CSS con *JavaScript*.
Questa tecnica, nota come *CSS-in-JS*, permette di definire e applicare stili personalizzati in modo modulare e dinamico per ciascun componente *React*, migliorando la leggibilità e la manutenibilità del codice. Grazie a *Styled Components*, è possibile creare componenti visivamente distinti, mantenendo il codice conciso e facilmente riutilizzabile.

La libreria si integra perfettamente con altre soluzioni di *styling*, come *Material-UI*, permettendo di personalizzare facilmente i componenti predefiniti di *Material-UI* tramite stili aggiuntivi o sovrascritti.
In questo modo, si può beneficiare della potenza di *Material-UI* per la creazione di interfacce utente moderne e responsive, mantenendo al contempo la flessibilità di *Styled Components* per adattare l'aspetto visivo ai requisiti specifici del progetto.

Node.js

Node.js è un ambiente di *runtime* per *JavaScript* che consente di eseguire codice *JavaScript* lato *server*, utilizzando il motore V8 di *Chrome*.
Questo permette di sviluppare applicazioni *server-side*, sfruttando le stesse competenze di *JavaScript* usate nel *Front-end*.
Il suo modello di programmazione asincrono e orientato agli eventi è ideale per gestire applicazioni scalabili e ad alte prestazioni, come *server web*, *Application Program Interface* ed applicazioni in tempo reale.

NestJS

Per la codifica del *Back-end* del progetto ho utilizzato *NestJs*, un *framework Back-end* costruito sulla base di *Node.js*, progettato per creare applicazioni *server-side* moderne, scalabili ed efficienti.
Esso si distingue per il suo approccio modulare e l'adozione di *TypeScript* come linguaggio principale, il che lo rende ideale per sviluppatori che cercano una struttura solida e una gestione ottimizzata del codice.
NestJS utilizza le potenzialità di *Node.js* per gestire applicazioni ad alte prestazioni e realizzare applicazioni *server-side* complesse, come *API RESTful*, applicazioni in tempo reale e microservizi.

Grazie al suo supporto nativo per *middleware*, guardie, filtri e *interceptor*, *NestJS* consente di implementare logiche complesse in modo chiaro e modulare.
Utilizza il modello asincrono e orientato agli eventi di *Node.js* per gestire in modo efficiente le richieste e le operazioni contemporanee, garantendo prestazioni elevate anche in contesti con carichi pesanti.

Le *API RESTful* sono un insieme di convenzioni e *best practices* per costruire interfacce di programmazione che permettano a *client* e *server* di comunicare attraverso il protocollo HTTP.
Esse si basano su risorse (ad esempio, dati o oggetti) che possono essere manipolate tramite operazioni standard di HTTP, come *GET* (per recuperare dati), *POST* (per creare nuovi dati), *PUT* (per aggiornare dati esistenti) e *DELETE* (per eliminare dati).
Grazie alla loro semplicità e scalabilità, le *API RESTful* sono ampiamente utilizzate nello sviluppo *web* e *mobile*.

Zod

Zod è una libreria *TypeScript* progettata per semplificare la validazione ed il *parsing* dei dati.
Consente di definire *schema* che specificano la forma ed i vincoli che i dati devono rispettare, permettendo di convalidare tipi di dati complessi come oggetti, stringhe, numeri, *array*.
Il principale punto forza di *Zod* risiede nella sua sintassi semplice e intuitiva, che permette di scrivere validazioni in modo dichiarativo e di catturare errori di tipo durante l'attività di sviluppo, evitando *bug* e problemi di coerenza nei dati.

Nel contesto dell'applicazione creata, ho utilizzato *Zod* sia nel *Front-end* che nel *Back-end* per garantire la correttezza dei dati.
Nel *Front-end*, *Zod* viene impiegato per la validazione dei dati di *login*, verificando che le credenziali inserite dall'utente siano nel formato corretto e rispettino determinati criteri (come la lunghezza minima della *password* o la validità dell'indirizzo *email*).
Questa validazione aiuta a migliorare l'esperienza utente e a ridurre gli errori lato *client*.

Nel *Back-end*, *Zod* viene utilizzato per validare le richieste ricevute dal *Front-end*, garantendo che i dati abbiano la struttura corretta e soddisfino le condizioni necessarie prima di essere elaborati ulteriormente.
L'uso di *Zod* nel *Back-end* riduce la possibilità di errori nei dati in ingresso, contribuendo ad una gestione più sicura ed affidabile delle informazioni, migliorando anche la robustezza e la manutenibilità del codice.

MongoDB

Come *database* ho utilizzato *MongoDB*, un *database NoSQL* altamente scalabile e flessibile, progettato per gestire dati sotto forma di documenti *JSON-like*.
Questo tipo di *database* è particolarmente indicato per applicazioni moderne che necessitano di una gestione efficiente di grandi volumi di dati non strutturati o semi-strutturati.
MongoDB si distingue per la sua capacità di scalare orizzontalmente, ovvero di espandersi facilmente su più *server* per gestire carichi di lavoro elevati, senza compromettere le prestazioni. È anche molto versatile nell'integrazione con diverse tecnologie e am-

bienti, rendendolo una scelta popolare per progetti che richiedono flessibilità e rapidità nello sviluppo.

Una delle principali differenze tra un *database SQL* (relazionale) e un *database NoSQL* come *MongoDB* riguarda la struttura dei dati.
Nei *database SQL*, i dati sono organizzati in tabelle con righe e colonne, ed ogni riga deve rispettare uno schema fisso predefinito.
Ciò rende i *database SQL* ideali per applicazioni con dati ben strutturati e con relazioni tra entità.
Tuttavia, questo approccio può risultare rigido quando si tratta di gestire dati complessi e non strutturati.

Al contrario, i *database NoSQL* come *MongoDB* non richiedono uno schema rigido.
I dati sono memorizzati in documenti che possono variare nel loro formato e struttura, il che consente una maggiore flessibilità.
Inoltre, i *database NoSQL* sono progettati per supportare operazioni di lettura e scrittura rapide su grandi volumi di dati, ed offrono una maggiore scalabilità orizzontale, il che significa che è più facile distribuire i dati su più *server* per gestire l'aumento delle richieste senza compromettere le prestazioni.
Questo li rende particolarmente adatti per applicazioni *web moderne* e scenari di microservizi.

La Figura 1.6 mostra uno schema contenente la differenza di struttura dei dati tra un *database SQL* ed un *database NoSQL*.

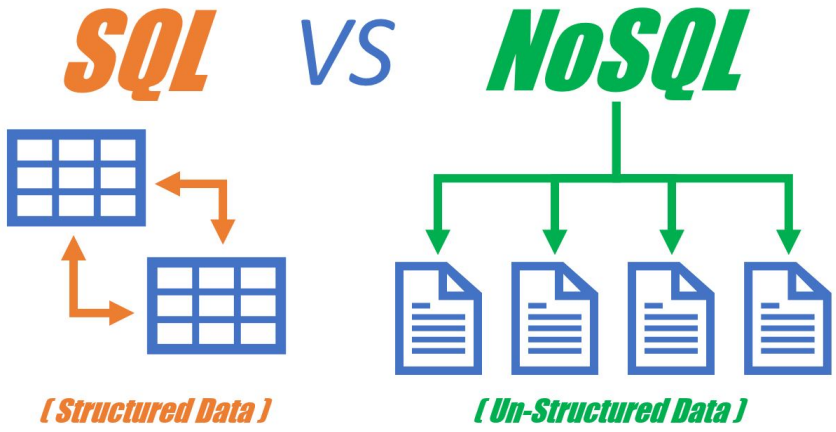


Figura 1.6: Differente strutturazione dei dati in un *database SQL* e in un *database NoSQL*
SQL vs NoSQL. URL: https://gowithcode.com/sql-vs-nosql

Mongoose

Mongoose è un *Object Data Modeling* (ODM) per *Node.js*, progettato per semplificare l'interazione tra *NestJS* e *MongoDB*.
Un ODM è una libreria che fornisce un'interfaccia tra un *database NoSQL* e il codice applicativo, consentendo agli sviluppatori di lavorare con oggetti *JavaScript* anziché interagire direttamente con il database.
In altre parole, un ODM come *Mongoose* trasforma i dati in formato *JSON-like* di *MongoDB* in oggetti *JavaScript* più facilmente gestibili, semplificando operazioni come la creazione, la lettura, l'aggiornamento e la cancellazione.

In *NestJS*, *Mongoose* viene utilizzato per definire *schema* di modelli, applicare validazioni e trasformazioni direttamente a livello di modello, e gestire *middleware* personalizzati per ogni operazione.
Questo permette agli sviluppatori di definire strutture dati robuste, convalidare le informazioni in ingresso e uscita, e applicare logiche direttamente sulle entità.
Mongoose supporta anche funzionalità avanzate come le *query* complesse, il popolamento di documenti (per integrare dati da più collezioni) e la gestione delle relazioni tra documenti, rendendo la gestione dei dati in *MongoDB* più intuitiva e scalabile all'interno di un'applicazione *NestJS*.

Puppeteer

Puppeteer è una libreria *Node.js* che fornisce un'interfaccia di alto livello per controllare un *browser Chromium* in modalità *headless*, cioè senza interfaccia grafica, per automatizzare attività come il rendering di pagine *web* e la generazione di *file* PDF.
Chromium è un *browser open-source* sviluppato da *Google*, che funge da base per browser come *Google Chrome, Microsft Edge, ecc.*

Grazie a *Puppeteer*, è possibile interagire programmaticamente con *Chromium*, simulando l'esecuzione di *script*, il caricamento di contenuti dinamici, l'interazione con il *DOM*, e la creazione di documenti.

Puppeteer è particolarmente utile per generare documenti PDF da template *web*, mantenendo la formattazione, gli stili e i *layout* originali, permettendo la personalizzazione di aspetti come margini, dimensioni della pagina e orientamento.
Questa libreria è stata impiegata nel progetto per generare *file* PDF contenenti i dati dei progetti generati, assicurando la coerenza e l'alta qualità dei documenti prodotti, sia in termini di formato che di contenuti.

HandleBars

Handlebars è una libreria *JavaScript* che facilita la creazione e popolazione di *template HTML* con dati dinamici.
Grazie alla sua sintassi semplice ed estensibile, *Handlebars* permette di separare la logica di presentazione dai dati, rendendo il codice più leggibile e manutenibile.
La libreria supporta l'uso di espressioni condizionali, cicli (*loops*) ed *helper* personalizzati, che permettono di manipolare e formattare i dati direttamente nel *template*.

Nel progetto, ho utilizzato *Handlebars* per popolare un *template HTML* con i dati dinamici relativi ai progetti.
Successivamente, questo *template* è stato passato a *Puppeteer* per generare un *file* PDF, mantenendo la formattazione e la struttura originale del *template*.
Questo approccio consente di ottenere documenti PDF coerenti e ben strutturati, basati su dati personalizzati, con una gestione separata tra la logica e la presentazione.

LangChain

LangChain è un *framework* avanzato progettato per facilitare lo sviluppo di applicazioni che sfruttano i LLM.
Gli LLM sono modelli di intelligenza artificiale addestrati su grandi quantità di dati testuali per comprendere e generare linguaggio naturale.
Grazie alla loro dimensione e complessità, possono eseguire compiti come traduzione, riassunti, completamento di frasi, e risposte a domande in modo altamente sofisticato.
Per una spiegazione più dettagliata sugli LLM si rimanda alla sezione §2.3.

LangChain semplifica l'interazione con modelli di linguaggio come quelli offerti da AWS *Bedrock*, fornendo strumenti per gestire chiamate API, orchestrare flussi di lavoro complessi, integrare memoria e combinare più modelli o fonti di dati in modo efficiente.
Nel progetto, ho utilizzato *LangChain* insieme ad AWS *Bedrock* per generare piani progettuali dettagliati.
Grazie alle sue capacità di orchestrazione, *LangChain* ha permesso di ottimizzare i flussi di interazione tra i modelli di linguaggio e i dati, garantendo una gestione fluida delle richieste e delle risposte.

LangChain include funzionalità avanzate come il *Prompt Engineering*, una disciplina dell'*Artificial Intelligence* (AI) che si occupa di progettare *prompt* ottimizzati per ottenere risposte precise e di alta qualità dai modelli di linguaggio.
Il *Prompt Engineering* consiste nel definire accuratamente la struttura e il contenuto del *prompt*, fornendo al modello un contesto chiaro e specifico per migliorare la pertinenza delle risposte.
Questa tecnica è fondamentale per massimizzare l'efficacia dei *prompt* e personalizzare i risultati in base alle esigenze applicative, rendendo l'interazione con i modelli più mirata ed efficiente.

Un'altra funzionalità avanzata di *LangChain* è la *Retrieval Augmented Generation* (RAG)[1].
Questa tecnica consente di migliorare le risposte dei modelli di linguaggio integrandole con una base di conoscenza esterna autorevole, al di fuori delle fonti di dati su cui i modelli sono stati addestrati.
La RAG permette di generare risposte più complesse e dettagliate, utilizzando documenti o *database* specifici per il problema di riferimento.
Sebbene questa funzionalità non sia stata applicata direttamente nel progetto, rappresenta un'opzione potente per scenari che richiedono risposte altamente personalizzate e affidabili.
LangChain, combinato con AWS *Bedrock*, rappresenta quindi uno strumento versatile per creare applicazioni scalabili, sfruttando appieno le potenzialità degli LLM avanzati.

[1] *Retrieval Augmented Generation*. URL: https://aws.amazon.com/it/what-is/retrieval-augmented-generation/.

La Figura 1.7 riporta le principali funzionalità di *LangChain*.

Figura 1.7: Pricipali funzionalità di *LangChain*
Key features of LangChain. URL:
https://datasciencedojo.com/blog/what-is-langchain/

AWS Amplify

Una piattaforma completa per lo sviluppo di applicazioni *web* e *mobile* integrata con servizi *cloud*. *AWS Amplify* fornisce strumenti per semplificare l'autenticazione degli utenti, la gestione delle API, lo *storage* di dati e l'*hosting* delle applicazioni.

Consente agli sviluppatori di connettersi rapidamente con altri servizi AWS, supportando la creazione di applicazioni scalabili e moderne grazie a funzionalità come il monitoraggio in tempo reale, l'integrazione con *framework* popolari ed il supporto per implementazioni *Continuous Integration/Continuous Deployment* (CI/CD).

CI/CD sono pratiche di sviluppo *software* che automatizzano l'integrazione continua del codice e il suo rilascio frequente.
Questi processi garantiscono maggiore efficienza, affidabilità e velocità nel ciclo di sviluppo.

AWS Cognito

Un servizio di gestione delle identità progettato per aggiungere funzionalità di autenticazione, registrazione e accesso federato alle applicazioni.
AWS Cognito facilita la gestione degli utenti, offrendo opzioni di autenticazione tramite *JSON Web Token* (JWT).

I JWT sono *token* standard utilizzati per trasferire in modo sicuro informazioni tra un *client* ed un *server*.
Essi contengono dati codificati (come l'identità dell'utente) e sono firmati digitalmente per garantire l'integrità e la sicurezza dei dati trasmessi.

Questi *token* sono ideali per applicazioni *web* moderne, poiché consentono una comunicazione senza necessità di sessioni persistenti sul *server*, migliorando le prestazioni e la scalabilità.

AWS Bedrock

Un servizio di *Generative AI* che permette agli sviluppatori di integrare modelli avanzati nelle proprie applicazioni senza dover possedere competenze specifiche in *Machine Learning*.

Il *Machine Learning* è una branca dell'intelligenza artificiale che si basa su algoritmi capaci di apprendere dai dati ed effettuare previsioni o prendere decisioni senza essere esplicitamente programmati.
La *Generative AI*, invece, rappresenta una sottocategoria dell'intelligenza artificiale focalizzata sulla creazione di contenuti nuovi e originali, come testo, immagini, musica o codice, basandosi su modelli pre-addestrati in grado di comprendere e replicare schemi presenti nei dati di addestramento.

Con AWS *Bedrock*, gli sviluppatori possono sfruttare modelli pre-addestrati forniti da aziende leader del settore, con possibilità di personalizzazione per adattarli ai propri casi d'uso specifici.
Questo servizio è ideale per sviluppare soluzioni innovative come *chatbot* avanzati, generazione di contenuti personalizzati, analisi approfondite di grandi volumi di dati e integrazione di funzionalità AI scalabili nelle applicazioni.

AWS S3 (*Simple Storage Service*)

Un servizio di archiviazione *cloud* progettato per memorizzare oggetti di qualsiasi dimensione con elevata scalabilità, disponibilità e sicurezza.
AWS S3 supporta il salvataggio di dati in forma di oggetti (*blob*) organizzati in *bucket*. È ideale per l'archiviazione e il recupero di file come immagini, video, documenti e *backup*.

Include funzionalità avanzate come:

- **Versionamento:** permette di gestire più versioni dello stesso *file*, mantenendo traccia delle modifiche e consentendo di ripristinare versioni precedenti se necessario;
- **Controllo degli accessi:** per proteggere i dati e regolarne l'accesso tramite politiche di sicurezza granulari;
- **Integrazione con altri servizi AWS:** per analisi, distribuzione di contenuti o *Machine Learning*.

Nel progetto specifico, ho utilizzato *AWS S3* per archiviare in modo sicuro i *file* PDF generati dinamicamente.
Grazie alla funzionalità di versionamento di *AWS S3*, ogni volta che un documento viene aggiornato o rigenerato, una nuova versione del *file* viene automaticamente salvata, mantenendo le versioni precedenti accessibili.
Questo processo garantisce che le modifiche ai documenti siano tracciate e che si

possieda una storicità completa dei *file*, permettendo di ripristinare versioni precedenti in caso di necessità, il tutto senza compromettere l'affidabilità o la sicurezza dei dati.

1.4.2 Tecnologie di supporto allo sviluppo

Node Package Manager (NPM)

NPM è il gestore ufficiale dei pacchetti per *Node.js*, utilizzato per installare, condividere e gestire librerie e moduli *JavaScript*.
Grazie a NPM, gli sviluppatori possono facilmente configurare e gestire le dipendenze necessarie per il proprio progetto, semplificando notevolmente il flusso di lavoro.

Nel contesto del mio progetto, ho utilizzato NPM per gestire le dipendenze sia del *Front-end* che del *Back-end*.
Per il *Front-end*, ha permesso di installare librerie come *React*, *Material-UI* ed altre dipendenze necessarie per lo sviluppo dell'interfaccia utente.
Per il *Back-end*, NPM ha facilitato l'installazione di pacchetti come *NestJS*, *Mongoose* e altre librerie essenziali per la gestione delle API e la comunicazione con il database *MongoDB*.

Utilizzando NPM, è stato possibile garantire che tutte le dipendenze fossero correttamente gestite e aggiornate, migliorando così l'efficienza e la manutenibilità del progetto.

Docker

Docker è una piattaforma che consente di creare, distribuire e gestire applicazioni tramite *container* leggeri e portatili.

I *container* consentono di isolare le applicazioni e le loro dipendenze, garantendo che l'ambiente di esecuzione sia consistente e riproducibile su qualsiasi sistema, che si tratti di sviluppo, *test* o produzione.
Questa tecnologia semplifica notevolmente il processo di *deployment*, risolvendo i problemi di compatibilità tra ambienti diversi.
I *container* offrono anche un'esecuzione più veloce e scalabile delle applicazioni.

Nel progetto ho utilizzato *Docker* per eseguire un'istanza di *MongoDB* in un *container*.
Questo approccio ha consentito di configurare facilmente un ambiente di *database* locale per lo sviluppo, mantenendo la stessa configurazione e versioni di *MongoDB* tra i vari ambienti, come quelli di *test* e produzione.
Grazie a *Docker*, ho potuto isolare *MongoDB* dal resto dell'applicazione, semplificando la gestione delle dipendenze e facilitando il processo di *deployment* in ambienti diversi.

Postman

Postman è uno strumento potente e intuitivo che consente di testare facilmente le API.
Permette di inviare richieste HTTP, come `GET`, `POST`, `PUT` e `DELETE`, e analizzare le risposte ricevute, semplificando notevolmente il processo di sviluppo e *debugging* delle API.
Grazie alla sua interfaccia grafica, *Postman* offre un modo semplice per definire i parametri delle richieste, visualizzare i risultati e monitorare il comportamento delle

API in tempo reale.

Nel progetto ho utilizzato *Postman* durante l'attività di codifica delle API per facilitare il *testing* delle varie funzionalità.
In particolare, è stato utilizzato per testare le API di creazione e gestione dei progetti, inviando richieste con i parametri appropriati e verificando che le risposte fossero corrette e conformi alle specifiche.
Questo approccio mi ha permesso di identificare rapidamente eventuali errori o problemi di funzionamento, migliorando l'affidabilità e la qualità delle API prima della loro integrazione nell'applicazione finale.

La Figura 1.8 mostra un esempio di utilizzo di *Postman* per testare una chiamata API, in particolare quella di generazione di un progetto.

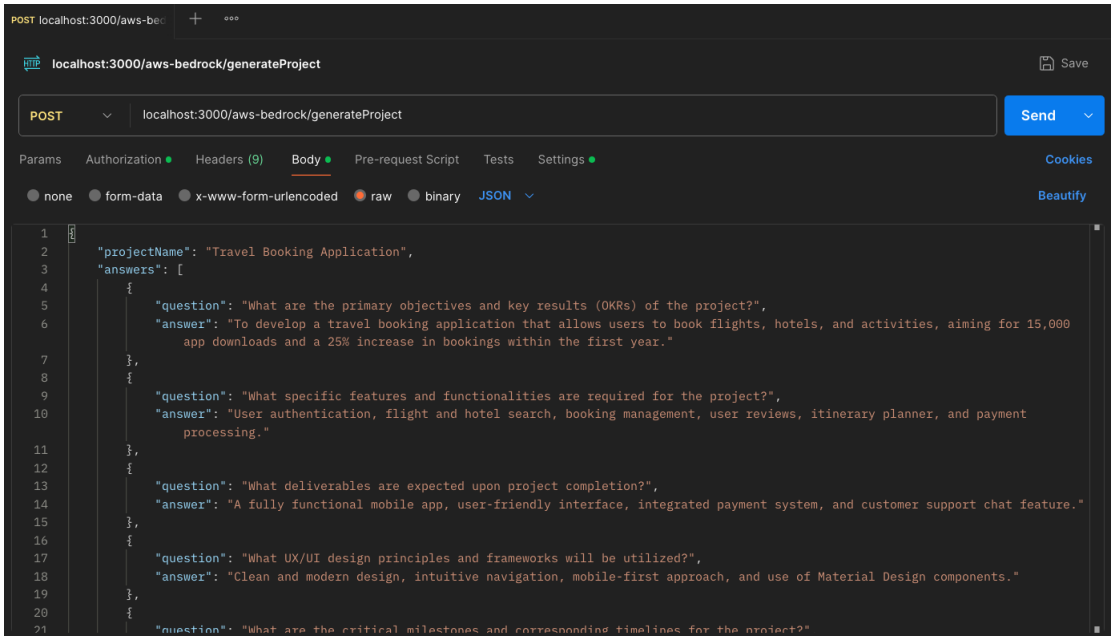


Figura 1.8: Esempio di utilizzo di *Postman* per chiamata API

StarUML

StarUML è uno strumento avanzato per la modellazione di diagrammi Unified Modeling Language (UML), un linguaggio standardizzato utilizzato per rappresentare visivamente le diverse componenti e interazioni di un sistema software.
UML fornisce un set di diagrammi che permettono di descrivere vari aspetti di un sistema, tra cui la struttura statica (come classi e oggetti) ed il comportamento dinamico (come interazioni e flussi di attività).
Tra i diagrammi UML più utilizzati ci sono i diagrammi dei casi d'uso, di sequenza e delle attività, che sono fondamentali per la progettazione e la documentazione di applicazioni.

Nel progetto ho utilizzato *StarUML* per creare i diagrammi dei casi d'uso, che rappresentano le interazioni tra gli attori (come gli utenti o i sistemi esterni) ed il sistema.

Questi diagrammi sono cruciali per descrivere le funzionalità offerte dal sistema e per evidenziare i vari scenari d'uso, facilitando la comprensione delle esigenze del progetto. I diagrammi dei casi d'uso creati con *StarUML* hanno semplificato la visualizzazione dei flussi operativi del sistema, migliorando la comunicazione tra il team di sviluppo e i clienti.

Git

Git è un sistema di controllo di versione distribuito progettato per tracciare le modifiche al codice sorgente e facilitare la collaborazione tra sviluppatori, rendendo la gestione di progetti *software* più efficiente e sicura.
Grazie alla sua natura distribuita, *Git* permette ad ogni sviluppatore di lavorare in modo indipendente, mantenendo una copia completa del progetto sul proprio ambiente locale, e successivamente sincronizzare le modifiche con un *repository* remoto.

Nel progetto ho utilizzato *Git* per versionare l'intero codice, garantendo che ogni modifica fosse tracciata e documentata.
È stato essenziale per la gestione delle diverse versioni del progetto, permettendo di lavorare su funzionalità parallele attraverso l'uso dei *branch*, e integrando le modifiche in modo sicuro tramite PR.
In questo modo, è stato possibile mantenere un flusso di lavoro ordinato e privo di conflitti, garantendo che il codice rimanesse sempre aggiornato e facilmente recuperabile in caso di necessità.

MongoDB Compass

MongoDB Compass è un'interfaccia grafica intuitiva che facilita l'interazione con il *database MongoDB*, permettendo di esplorare e modificare facilmente i dati, visualizzare gli indici, eseguire *query*, e gestire il database in modo efficiente.
Grazie alla sua interfaccia visiva, semplifica operazioni complesse come la gestione della struttura dei dati, l'analisi delle prestazioni delle *query*, e la visualizzazione dei documenti.

Nel progetto ho utilizzato *MongoDB Compass* per accedere facilmente al database *MongoDB*, consentendo di visualizzare, modificare e analizzare i dati durante l'attività di sviluppo.
È stato uno strumento fondamentale per esplorare le collezioni, controllare i dati inseriti, semplificando il processo di *debugging* e ottimizzazione delle *query*.
In questo modo, mi è stato possibile gestire e monitorare il *database* con maggiore efficienza, garantendo un flusso di lavoro più rapido e preciso.

La Figura 1.9 mostra un esempio di utilizzo di *MongoDB Compass* per visualizzare un documento all'interno di una collezione.

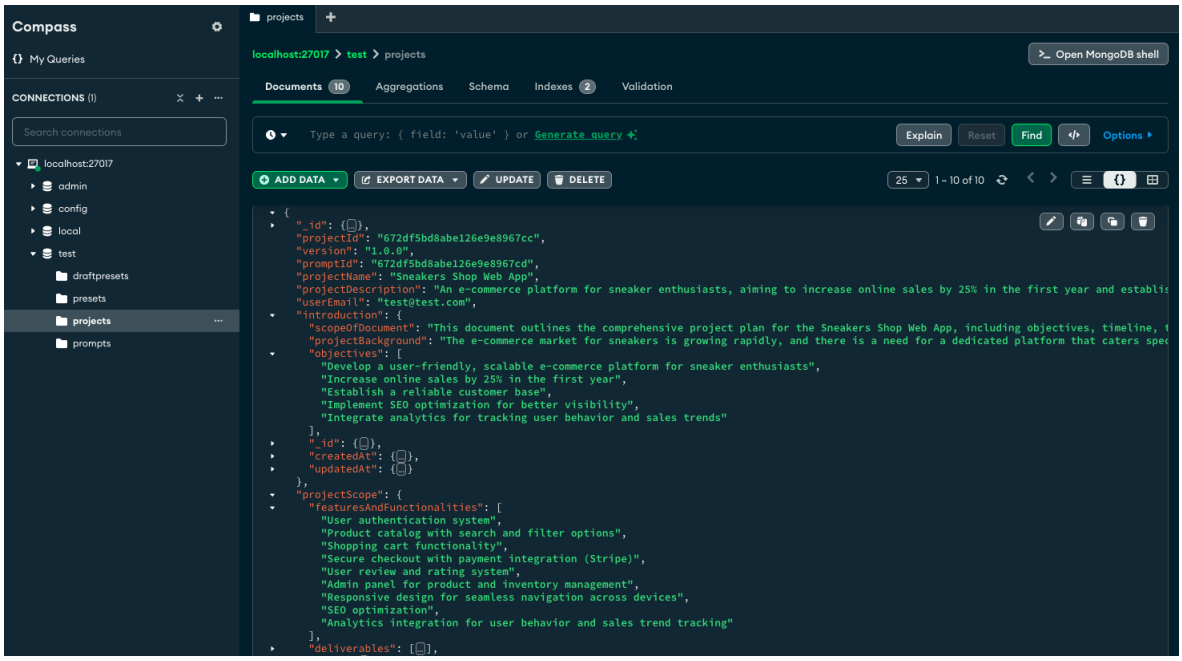


Figura 1.9: Visualizzazione di un documento in *MongoDB Compass*

1.4.3 Tecnologie di verifica e validazione del codice

Jest

Jest è un framework di *testing JavaScript* progettato per garantire la qualità del codice, supportando *test* unitari e di integrazione.
Grazie alla sua semplicità, velocità e alle funzionalità avanzate come la simulazione di funzioni e la generazione di *report* dettagliati, *Jest* è particolarmente adatto per l'automazione dei *test* nei progetti moderni.

Nel progetto ho utilizzato *Jest* per creare *test* unitari sia per il *Front-end* che per il *Back-end*, permettendo di verificare il corretto funzionamento dei singoli componenti del sistema.
Per il *Front-end*, *Jest* è stato impiegato per testare componenti *React* e logiche *JavaScript*, mentre per il *Back-end* è stato utilizzato per testare le API e i servizi implementati in *NestJS*.
L'uso di *Jest* ha facilitato il rilevamento tempestivo di errori e ha migliorato l'affidabilità del codice, garantendo una qualità elevata in tutte le fasi di sviluppo.

React Testing Library

React Testing Library è una libreria per testare i componenti *React*, progettata per favorire il *testing* dell'interazione dell'utente piuttosto che l'implementazione interna del componente.
Si concentra sul comportamento dell'interfaccia utente e sulla corretta gestione degli eventi, consentendo di simulare azioni dell'utente come clic, inserimento di testo e interazioni con i *form*.

Nel progetto ho utilizzato *React Testing Library* per *testare* i componenti *React*, garantendo che ciascun componente rispondesse correttamente alle azioni dell'utente e alle modifiche di stato.
In particolare, è stato impiegato per verificare la corretta visualizzazione dei dati, la gestione degli eventi e l'integrazione tra i vari componenti dell'interfaccia utente.
L'approccio focalizzato sull'interazione utente ha contribuito a testare la funzionalità complessiva dell'applicazione, migliorando la qualità e l'affidabilità dell'esperienza utente.

Prettier

Prettier è uno strumento di formattazione automatica del codice che aiuta a mantenere uno stile uniforme e coerente, migliorando la leggibilità e la manutenzione del codice nel lungo periodo.
Supporta una vasta gamma di linguaggi di programmazione e può essere facilmente integrato nei flussi di lavoro di sviluppo.

Nel progetto ho utilizzato *Prettier* per formattare automaticamente il codice *TypeScript*, garantendo che tutte le modifiche apportate al codice rispettassero uno stile comune e standardizzato.
Questa integrazione ha semplificato la gestione del codice, migliorando la coerenza e riducendo la possibilità di errori legati a incongruenze stilistiche.
Inoltre, *Prettier* è stato integrato nei flussi di lavoro di sviluppo, assicurando che il codice formattato correttamente venisse applicato in modo continuo, anche durante l'attività di revisione e aggiornamento del codice.

ESLint

ESLint è uno strumento di *linting* per *JavaScript* che aiuta a individuare e correggere errori nel codice, migliorando la qualità e la coerenza del codice stesso.
Supporta configurazioni personalizzabili e si integra facilmente con IDE, offrendo un'esperienza di sviluppo più fluida.

Nel progetto ho utilizzato *ESLint* con *TypeScript* per monitorare e mantenere alta la qualità del codice.
La configurazione di *ESLint* per *TypeScript* consente di identificare potenziali errori, migliorare la coerenza tra le diverse parti del codice e applicare regole stilistiche personalizzabili.
Questo strumento ha contribuito a prevenire bug, ottimizzare la struttura del codice e applicando le *best practices* nel trattamento del codice *TypeScript*.

1.5 Propensione aziendale all'innovazione

L'azienda Zero12 è profondamente orientata all'innovazione e alla sperimentazione di nuove tecnologie.
In qualità di partner di AWS, la società si distingue per l'interesse nella ricerca e nello sviluppo nel campo dei servizi *cloud*.

Durante il mio periodo di *stage*, diversi membri dell'azienda hanno partecipato a conferenze in Emilia Romagna, organizzate presso la sede di UAN Company, acquisitrice di Zero12, per esplorare nuove tecnologie e come queste possano essere integrate nei progetti futuri dell'azienda.
Mi è stato riferito che tali conferenze e corsi di formazione sono eventi frequenti, in quanto Zero12 promuove attivamente la formazione continua per tutti i suoi dipendenti, riflettendo il suo impegno per il miglioramento e l'evoluzione costante delle competenze interne.

Capitolo 2

Progetto di *stage*

2.1 Gestione aziendale degli *stage*

Zero12 considera gli *stage* con gli studenti dell'Università di Padova un'opportunità molto importante per formare nuove risorse umane, per far conoscere l'azienda e per valutare possibili nuove tecnologie.
Grazie all'evento *STAGE-IT*, svolto ad aprile 2024 e promosso da Confindustria Veneto Est e dall'Università di Padova, l'azienda ha avuto l'opportunità di presentarsi agli studenti e di esporre i propri progetti di *stage*.

Gli *stage* permettono agli studenti di applicare le conoscenze acquisite durante il corso di studi, di confrontarsi con un ambiente lavorativo reale e di sviluppare competenze professionali.
Per l'azienda, questi progetti rappresentano un ottimo strumento per sperimentare nuove idee da presentare a potenziali clienti, oltre che per formare future risorse umane. Questo è dimostrato dal fatto che molti attuali dipendenti sono stati assunti dopo aver svolto uno *stage* presso l'azienda.
Tutti i colleghi sono molto disponibili e pronti ad aiutare, al fine di rendere la permanenza dello stagista il più produttiva e piacevole possibile.

2.2 Descrizione proposta di *stage*

L'idea alla base dello *stage* è quella di automatizzare la creazione di un documento denominato `Allegato Tecnico`.
Questo documento raccoglie tutte le informazioni tecniche relative ad un progetto *software*, comprese le tecnologie utilizzate, la struttura del progetto, le scelte architetturali ed il piano di *testing*, per citare alcune delle sezioni.
Fino ad oggi, l'azienda ha sempre realizzato questo documento manualmente.
Tuttavia, con l'aumento dei progetti e delle risorse impiegate, l'introduzione di un *tool* per automatizzare questo processo, anche solo parzialmente, risulta particolarmente interessante.

L'obiettivo dello *stage* è quindi la realizzazione di una piattaforma *web* che, tramite l'uso di sistemi di *Generative AI*, permetta di generare automaticamente questo documento.
L'utente finale della piattaforma sarà un membro del *team* di sviluppo od il *project-manager*.
Questi andrà a compilare dei *Preset*, ovvero una serie di domande create *ad hoc* per raccogliere le informazioni necessarie alla creazione del documento, ottimizzate in base al tipo di progetto da sviluppare (ad esempio, una *Web App* o una *Mobile App*).
Il sistema utilizzerà i servizi di *Generative AI* forniti da AWS per creare il documento automaticamente.

Una volta generato, l'utente avrà la possibilità di rigenerare il documento, in parte o nella sua totalità, inserendo un *Prompt*.
Questo *Prompt* sarà una frase o un paragrafo che guiderà il sistema, consentendo all'utente di specificare le migliorie o modifiche da apportare al documento.
Tutte le versioni del documento verranno salvate, permettendo di visualizzare le differenze tra una versione e l'altra.
Una volta soddisfatto del risultato, l'utente potrà scaricare il documento in formato *PDF*.
La Figura 2.1 mostra uno schema che rappresenta l'idea alla base del progetto.

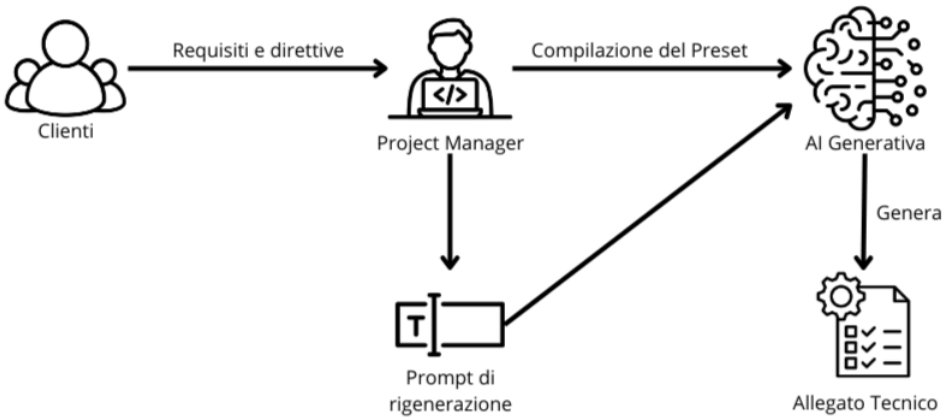


Figura 2.1: Schema del progetto

2.3 *Large Language Models*

2.3.1 Introduzione

I *Large Language Model* sono modelli computazionali progettati per interpretare e generare testo in linguaggio naturale[1].
Sebbene le loro basi teoriche risalgano alla "Teoria dell'apprendimento statistico" sviluppata tra gli anni '80 e '90, è solo negli ultimi anni che hanno guadagnato un'attenzione globale significativa.
Questo è stato possibile grazie alla crescente disponibilità di dati e di potenza di calcolo, che hanno permesso la creazione di modelli sempre più grandi e complessi.
Le applicazioni dei *Large Language Model* sono molteplici: traduzioni automatiche, generazione di testi, riconoscimento vocale, e *chatbot* sono solo alcuni esempi.

Il principio di funzionamento di un *Large Language Model* si basa sull'addestramento su vastissime quantità di dati testuali, come libri, articoli *web* e altre risorse scritte, per apprendere la struttura e il significato del linguaggio naturale.
Successivamente, questi modelli sono in grado di generare testo coerente e sensato in base al contesto fornito.

I *Large Language Model* utilizzano un approccio "probabilistico", ovvero cercano di prevedere la parola successiva in una sequenza testuale basandosi sulle parole precedenti.
Ogni parola ha una probabilità associata, calcolata dal modello.
La parola con la probabilità più alta viene spesso selezionata per garantire coerenza, ma ciò può portare a risultati prevedibili e poco creativi, che possono risultare poco interessanti per l'utente.

Per ovviare a questo problema, si utilizzano i seguenti parametri[2]:

- **Temperatura**: regola la casualità delle scelte del modello.
  Un valore basso predilige la scelta della parola con probabilità più alta, mentre un valore alto aumenta la casualità delle scelte.
- ***Top-K***: limita la scelta alle *top-K* parole con probabilità più alta.
  Questo permette di evitare scelte troppo casuali e poco coerenti, andando a bilanciare prevedibilità e casualità.

Mostriamo ora un esempio: supponiamo che una *Large Language Model* stia generando una frase a partire dalla seguente sequenza di parole:

"Il sistema ha ricevuto una richiesta da"

Le probabilità per le parole successive potrebbero essere le seguenti:

- *"un utente autenticato"*: 0.4
- *"un client esterno"*: 0.3
- *"un microservizio interno"*: 0.2
- *"un server remoto"*: 0.1

[1] *Introduction to LLM.* URL: https://www.hpe.com/it/it/what-is/large-language-model.html.
[2] *LLM parameters.* URL: https://ivibudh.medium.com/a-guide-to-controlling-llm-model-output-exploring-top-k-top-p-and-temperature-parameters-ed6a31313910.

1. Senza utilizzo dei parametri, il modello seleziona la parola con probabilità più alta:
   "Il sistema ha ricevuto una richiesta da ***un utente autenticato****"*;

2. Con temperatura alta (ad esempio 1.0), il modello seleziona una parola in modo casuale, ad esempio:
   "Il sistema ha ricevuto una richiesta da ***un server remoto****"*;

3. Con *top-K* basso (ad esempio 2), il modello seleziona una parola tra le due con probabilità più alta, ad esempio:
   "Il sistema ha ricevuto una richiesta da ***un client esterno****"*.

Un altro concetto fondamentale per comprendere il funzionamento dei *Large Language Model* è quello dei *token*.
Un *token* rappresenta la più piccola unità di testo che un LLM può elaborare.
I *token* possono essere parole intere, parti di parole o anche singoli caratteri, a seconda del modello e della sua configurazione.

La suddivisione in *token* è cruciale per l'addestramento dei *Large Language Model*, poiché il modello deve essere in grado di gestire e prevedere sequenze di *token* piuttosto che intere frasi o parole.
Durante l'addestramento, il modello apprende le probabilità di transizione tra i *token*, costruendo una rappresentazione statistica del linguaggio naturale.

2.3.2 Confronto tra i principali LLM

Di seguito è presente un confronto tra i principali LLM attualmente disponibili sul mercato[3], con l'obiettivo di evidenziare le loro differenze, i casi d'uso principali, i loro punti di forza ed i limiti di ciascun modello.
Questi modelli sono accessibili tramite *AWS Bedrock*[4], ad eccezione di *GPT-4*, incluso comunque nel confronto per la sua rilevanza e diffusione.

GPT-4 (OpenAI)

Descrizione: *GPT-4* è un modello all'avanguardia progettato per risolvere compiti complessi grazie alla sua elevata capacità di comprensione e generazione del linguaggio. Anche se non disponibile tramite *AWS Bedrock*, è una delle soluzioni più utilizzate per la sua accuratezza e versatilità.

Casi d'uso:

- **Generazione tecnica:** Creazione di documentazione tecnica dettagliata;
- **Analisi avanzata:** Interpretazione di dati e contenuti complessi;
- **Supporto per sviluppatori:** *Debugging* e generazione di codice ottimizzato;
- **Ricerca scientifica:** Sintesi e analisi di *paper* scientifici.

[3] *Key differences between main LLMs.* URL: https://ai-pro.org/learn-ai/articles/a-comprehensive-comparison-of-all-llms/.
[4] *AWS Bedrock.* URL: https://aws.amazon.com/it/bedrock/.

Punti di forza:

- **Capacità di contesto estesa:** Supporta fino a 32k *token*, adatto per documenti complessi;
- **Versatilità *multi-task*:** Funziona bene su domande aperte e contesti tecnici specifici;
- ***Fine-tuning* implicito:** Non necessita di *tuning* per molti casi d'uso, grazie ad un addestramento generale molto efficace.

Limiti:

- **Costo elevato:** Il prezzo per *token* è più alto rispetto ad altre opzioni;
- **Latenza:** *Input* complessi possono rallentare il tempo di risposta.
- **Non nativo per *AWS Bedrock*:** Necessita di integrazione tramite API *OpenAI*.

Claude 3.5 Sonnet (Anthropic)

Descrizione: *Claude 3.5 Sonnet* è ottimizzato per la gestione di contesti estesi e la generazione di risposte sicure e trasparenti.
Su *AWS Bedrock*, è utilizzato per applicazioni con grandi volumi di testo e analisi linguistiche.

Casi d'uso:

- **Analisi documentale:** Generazione e sintesi di documenti di grandi dimensioni (fino a 100k *token*);
- **Automazione *HR* e legale:** Risposte standardizzate e conformità normativa;
- **Interfacce conversazionali:** *Chatbot* sicuri per il supporto clienti ed interno.

Punti di forza:

- **Capacità di contesto estesa:** Supporta *input* fino a 100k *token*, ideale per documenti lunghi;
- **Bias ridotto:** Allenato per minimizzare risposte non sicure o inappropriate;
- **Costi contenuti:** Efficiente rapporto costo-*token*.

Limiti:

- **Latenza:** *Input* complessi possono rallentare il tempo di risposta;
- **Risposte conservative:** Tende a evitare *output* creativi o fuori dall'ordinario.
- **Limitato per codice:** Meno supporto per generazione di codice avanzato.

Amazon Titan

Descrizione: *Titan* è il modello di intelligenza artificiale sviluppato da *Amazon*, ottimizzato per l'integrazione nativa con *AWS Bedrock* ed altre soluzioni AWS.
Si concentra su casi d'uso aziendali pratici e robusti.

Casi d'uso:

- **Elaborazione dati strutturati:** Supporto per analisi di *report* aziendali e *database*;
- ***Chatbot* aziendali:** Personalizzabili per specifici settori industriali;
- **Supporto operativo:** Automazione di flussi di lavoro e riepiloghi aziendali.

Punti di forza:

- **Integrazione nativa con AWS:** Compatibilità totale con strumenti AWS come *DynamoDB* e *S3*;
- **Ottimizzazione della sicurezza:** Protezione dei dati aziendali con politiche AWS rigorose;
- **Efficienza nei costi:** Disegnato per fornire prestazioni elevate anche con *budget* contenuti.

Limiti:

- **Capacità di linguaggio generico inferiore:** Non raggiunge la qualità linguistica di *GPT-4* o *Claude 3.5*;
- **Meno innovativo:** Adatto a compiti definiti, non eccelle in creatività o flessibilità;
- **Limitata flessibilità:** Richiede configurazioni specifiche per settori diversi.

Llama (Meta)

Descrizione: *Llama* è un modello *open-source* progettato per essere altamente personalizzabile, ideale per casi in cui è richiesto un controllo completo sul *fine-tuning*.
Su *AWS Bedrock*, è particolarmente utile per applicazioni che richiedono soluzioni specifiche per il dominio applicativo.

Casi d'uso:

- **Addestramento personalizzato:** Modelli ottimizzati per domini ristretti come finanza o medicina;
- **Soluzioni locali:** Applicazioni in ambienti chiusi per garantire massima *privacy*;
- **Sperimentazione:** Perfetto per ricerca accademica e/o sviluppo di prototipi.

Punti di forza:

- ***Open-source*:** Offre pieno controllo per modifiche e ottimizzazioni;
- **Scalabilità flessibile:** Può essere ridimensionato per applicazioni *on-premises* o *cloud*.

Limiti:

- **Richiede competenze tecniche avanzate:** Il *tuning* e l'implementazione necessitano di risorse specializzate;
- **Prestazioni inferiori senza *tuning*:** Il modello base è meno performante rispetto a *GPT-4* o *Claude 3.5*;
- **Ambiente di sviluppo più complesso:** Meno pronto all'uso rispetto ai modelli pre-addestrati.

Di seguito una tabella che mostra un confronto riassuntivo tra i modelli presentati:

Modello	Casi d'uso	Punti di forza	Limiti
GPT-4	Automazione avanzata, analisi tecnica	Supporta 32k *token*, versatile e accurato	Costoso, non nativo su *AWS Bedrock*
Claude 3.5	Generazione documenti lunghi, chatbot sicuri	Contesto ampio, sicurezza nei dati	Risposte conservative, latenza elevata
Amazon Titan	Report aziendali, automazione	Integrazione nativa AWS, costi ridotti	Meno versatile, output linguistico generico
Llama	Addestramento personalizzato	*Open-source*, flessibilità	Prestazioni limitate senza modifiche avanzate

Tabella 2.1: Tabella confronto riassuntivo *LLM*

2.4 Obiettivi aziendali

Gli obiettivi forniti dall'azienda sono stati definiti a priori dell'inizio dello stage all'interno del documento del `Piano di Lavoro`, essi si suddividono in:

- **OO: Obiettivi obbligatori**, sono gli obiettivi minimi che devono essere raggiunti per considerare lo stage completato con successo;
- **OD: Obiettivi desiderabili**, sono gli obiettivi non necessari, ma che vanno ad aggiungere valore al prodotto finito;
- **OF: Obiettivi facoltativi**, sono gli obiettivi opzionali, non necessari per il completamento del progetto.

All'interno della seguente tabella sono elencati gli obiettivi aziendali dello stage, aventi la seguente nomeclatura:

R-N

Dove:

- **R** $\in \{OO, OD, OF\}$ rappresentante la tipologia di obiettivo;
- **N** $\in N$ rappresentante il numero progressivo dell'obiettivo.

Codice Obiettivo	Descrizione Obiettivo
Obiettivi Obbligatori	
OO-1	**Apprendimento delle tecnologie di sviluppo:** Acquisire competenze pratiche nell'uso di *React*, *NestJS* e *MongoDB* per la progettazione e lo sviluppo di applicazioni.
OO-2	**Gestione del versionamento del codice:** Apprendere l'uso di *Git* e adottare *Git Flow* come metodologia per il controllo delle versioni e la collaborazione.
OO-3	**Analisi e scelta del LLM:** Valutare i modelli disponibili per selezionare quello più adatto al progetto.
OO-4	**Introduzione alle metodologie agili:** Familiarizzare con le metodologie agili di sviluppo per la gestione efficace di progetti.
OO-5	**Pianificazione e gestione giornaliera:** Imparare a gestire *task* e obiettivi giornalieri allineati al piano di lavoro.
OO-6	**Sviluppo di una *web app*:** Progettare e realizzare un'applicazione *web* per consentire l'interazione dell'utente con la piattaforma.
OO-7	**Sviluppo ed integrazione con *Generative AI*:** Implementare i flussi logici del progetto e integrare i servizi di *Generative AI* scelti.

Codice Obiettivo	Descrizione Obiettivo
OO-8	**Documentazione API:** Creare una documentazione *Swagger* per le API sviluppate.
OO-9	**Documento di analisi progettuale:** Redigere un documento tecnico che descriva l'architettura e le componenti principali della piattaforma.
OO-10	***User Story Mapping:*** Realizzare una mappatura delle *User Stories* per descrivere e organizzare i requisiti del progetto.
Obiettivi Desiderabili	
OD-1	**Comparazione tra modelli LLM:** Effettuare un'analisi comparativa tra almeno due LLM per verificarne le differenze in termini di prestazioni, funzionalità e costi.
Obiettivi Facoltativi	
OF-1	**Piattaforma di amministrazione:** Creare una piattaforma *admin* per la gestione dei *Preset* e dei modelli LLM.

Tabella 2.2: Obiettivi aziendali dello *stage*

2.5 Vincoli

2.5.1 Vincoli tecnologici

L'azienda ha definito una serie di vincoli tecnologici per la realizzazione del progetto, al fine di garantire standard qualitativi elevati e l'adozione di pratiche consolidate. Questi vincoli sono essenziali per assicurare l'integrazione con l'ecosistema tecnologico esistente e per sfruttare al meglio le competenze aziendali.

Di seguito vengono descritti nel dettaglio:

- **Documentazione *Swagger* delle API:** Ogni API sviluppata dovrà essere accompagnata da una documentazione completa generata con *Swagger*.
  Questo strumento consente di definire e visualizzare la struttura delle *API RESTful*, garantendo una comunicazione chiara e standardizzata tra i diversi *team* di sviluppo;
- ***Test* di unità:** È necessario implementare *test* di unità per tutte le componenti del sistema, sia *Front-end* che *Back-end*, al fine di garantire la correttezza funzionale del codice e ridurre i rischi di regressioni durante le fasi di sviluppo e manutenzione;
- **Ambiente di *test* funzionante:** La *web application* deve essere operativa in un ambiente di *test* che simuli condizioni reali.
  Questo è fondamentale per individuare eventuali problematiche prima del rilascio in produzione;
- **Tecnologie *Front-end*:** Nel *Front-end* è obbligatorio l'utilizzo di *React.js*, un *framework JavaScript* moderno che garantisce alta modularità e performance, e della libreria *Material-UI* per la progettazione di interfacce utente coerenti e responsivi.
  Questi strumenti permettono di sviluppare applicazioni altamente interattive con una *user experience* avanzata;
- **Autenticazione:** Per la gestione dell'autenticazione e autorizzazione, devono essere utilizzati *AWS Amplify* e *AWS Cognito*, soluzioni che offrono sicurezza, scalabilità e facilità di integrazione con altre piattaforme AWS.
- **Tecnologie *Back-end*:** Nel *Back-end*, lo sviluppo deve avvenire utilizzando *NestJS*, un framework basato su *Node.js* che favorisce la creazione di architetture scalabili e modulari.
  È inoltre obbligatorio andare ad implementare *API RESTful*, aderendo ai principi *REST* per garantire interoperabilità e riusabilità dei servizi;
- **Gestione dei documenti:** La generazione dei documenti deve essere effettuata utilizzando *AWS Bedrock*, una piattaforma innovativa che sfrutta modelli di AI per la creazione e l'elaborazione automatizzata di contenuti altamente personalizzati.
  I documenti generati devono essere archiviati in modo sicuro nel *cloud* tramite *AWS S3*, che garantisce alta durabilità, scalabilità e accessibilità globale;

- ***Database*:** Come *database* per il progetto è richiesto l'uso di *MongoDB*, un *database NoSQL* scalabile e flessibile che consente di memorizzare dati in formato *JSON-like*.
  Questa scelta permette di gestire facilmente dati non strutturati o semi-strutturati e garantisce alte prestazioni per le operazioni di lettura e scrittura.

Questi vincoli tecnologici rappresentano le linee guida principali per lo sviluppo del progetto, promuovendo l'uso di tecnologie all'avanguardia e l'adozione di metodologie che assicurano la qualità e l'efficienza del prodotto finale.

2.5.2 Vincoli organizzativi

L'organizzazione del lavoro ha seguito precise linee guida per garantire un'efficace gestione del progetto ed una comunicazione fluida tra i vari attori coinvolti.

Di seguito vengono illustrati i vincoli organizzativi imposti dall'azienda:

- **Creazione di *User Stories*:** Durante l'attività di analisi, è necessario andare a creare *User Stories* che descrivano i requisiti del sistema dal punto di vista dell'utente.
  Ogni *user story* deve includere una descrizione, i criteri di accettazione ed una priorità. Tali *user stories* devono essere tracciate su *Jira*.
  Per una descrizione più dettagliata delle *User Stories*, si rimanda alla sezione §3.2.1;

- **Comunicazione interna tramite *Slack*:** Per le interazioni con il *tutor* aziendale e altre figure aziendali, è richiesto l'uso di *Slack* come strumento di comunicazione.
  La piattaforma consente di organizzare discussioni in canali tematici e mantenere una cronologia facilmente accessibile;

- **Lavoro a *sprint*:** Il lavoro è suddiviso in *sprint*, ovvero brevi cicli di sviluppo della durata di una settimana.
  Uno *sprint* rappresenta un periodo di tempo delimitato durante il quale il *team* lavora per completare un insieme di obiettivi predefiniti, spesso basati su specifiche *user stories*.
  Ogni *sprint* inizia con una pianificazione, dove vengono definiti i compiti prioritari da completare, e termina con una *sprint retrospective*, una riunione durante la quale il *team* analizza i risultati ottenuti, identifica eventuali problematiche e propone soluzioni per migliorare i successivi *sprint*.
  Questo approccio garantisce un processo di sviluppo iterativo e incrementale, promuovendo la consegna continua di valore;

- **Utilizzo di *Git Flow*:** Durante lo stage, il lavoro di sviluppo sarà organizzato utilizzando *Git Flow*, una metodologia di gestione del flusso di lavoro in *Git* che aiuta a strutturare in modo efficiente i processi di sviluppo e gestione del codice sorgente[5].
  Git Flow si basa su un ramo principale (`main`) e due rami di sviluppo principali: `develop` per lo sviluppo quotidiano e `feature` per le nuove funzionalità.

[5] *Git Flow*. URL: https://www.atlassian.com/git/tutorials/comparing-workflows/gitflow-workflow.

Inoltre, vengono utilizzati rami di `release` per preparare nuove versioni stabili del *software* e rami di `hotfix` per correggere errori urgenti nelle versioni di produzione. Il flusso di lavoro con *Git Flow* permette di isolare le modifiche, garantire che il codice sia sempre testato e pronto per essere distribuito, e facilitare il lavoro in *team*, mantenendo il codice principale sempre stabile.
Ogni modifica al codice viene effettuata su rami specifici, evitando conflitti e semplificando il processo di fusione (*merge*) delle modifiche;

- **Comunicazione costante con il *tutor* aziendale:** Sono previsti *meeting* settimanali con il *tutor* aziendale per effettuare la *sprint retrospective*, discutere i progressi ed affrontare eventuali problematiche.
  Qualsiasi difficoltà o dubbio deve essere comunicato tempestivamente per garantire un supporto immediato e risolvere le criticità in modo efficace;

- **Comunicazione costante con il Relatore:** È richiesta una comunicazione regolare con il relatore ogni cinque giorni lavorativi per fornire aggiornamenti sullo stato di avanzamento, verificare il rispetto del Piano di Lavoro e dare una pianificazione riguardo al periodo successivo.

Questi vincoli organizzativi sono stati definiti per facilitare la pianificazione, il monitoraggio e l'esecuzione del progetto, promuovendo un ambiente di lavoro ben strutturato.

2.5.3 Vincoli alla progettazione e a alla codifica

La scelta delle tecnologie utilizzate nel progetto è stata in gran parte imposta dall'azienda.
In particolare, l'azienda ha imposto l'adozione di *ReactJS* come *framework* per il *Front-end*, di *NestJS* per il *Back-end* e di *MongoDB* come *database*.
Queste scelte sono state motivate da una serie di considerazioni interne, tra cui la familiarità aziendale con le tecnologie e la loro capacità di soddisfare le esigenze di scalabilità e manutenibilità del progetto.

Inoltre, l'azienda ha imposto l'utilizzo di una serie di servizi AWS per la gestione dell'autenticazione, la generazione dei progetti e il salvataggio dei documenti.
Questi sono stati ampiamente discussi nella Sezione 1.4 relativa alle tecnologie di sviluppo.

2.5.4 Scelta del LLM

La scelta del LLM più adatto per la generazione dei progetti è stato un passo cruciale nel processo di sviluppo, poiché il modello scelto avrebbe influenzato significativamente la qualità e la coerenza dei contenuti generati.
Durante l'attività di valutazione, ho considerato diversi modelli di linguaggio avanzati, tra cui *GPT-4*, *Claude 3.5 Sonnet*, *Amazon Titan* e *Llama*, ognuno dei quali dettagliamente descritto nella Sezione 2.3.2.

Ho preso la decisione finale tenendo conto delle specifiche necessità del progetto, in particolare nell'ambito dell'utilizzo di *AWS Bedrock*, la piattaforma scelta per l'integrazione con i modelli di linguaggio.

Esclusione di GPT-4

Ho scartato a priori il modello *GPT-4*, pur essendo uno dei più potenti e avanzati modelli di linguaggio disponibili.
Questa scelta è stata dettata dal fatto che *GPT-4* non fosse disponibile tramite le API di *AWS Bedrock*, la piattaforma utilizzata per la gestione dei modelli nel progetto.
Poiché l'architettura del sistema dipendeva fortemente dall'integrazione con *AWS Bedrock*, l'impossibilità di utilizzare *GPT-4* ha automaticamente escluso questo modello dalla selezione.

Scelta di *Claude 3.5 Sonnet*

Dopo aver escluso *GPT-4*, la mia scelta si è concentrata sui tre LLM rimanenti: *Claude 3.5 Sonnet*, *Amazon Titan* e *Llama*.
Dopo una serie di valutazioni, discussioni con il *tutor* aziendale e approfondimenti sulle caratteristiche dei modelli, la mia decisione finale è ricaduta su *Claude 3.5 Sonnet* di *Anthropic*.

La scelta di *Claude 3.5 Sonnet* è stata motivata principalmente dalla sua capacità di generare testi molto lunghi, una caratteristica fondamentale per il progetto, che prevedeva la generazione di progetti molto dettagliati.
A differenza di altri modelli, che possono essere più efficienti nel generare testi brevi e concisi, *Claude 3.5 Sonnet* ha dimostrato di essere in grado di mantenere una coerenza narrativa e una logica fluida anche su testi di grande lunghezza.

Uno dei principali svantaggi in merito alla scelta di *Claude 3.5 Sonnet* è la latenza maggiore che esso comporta, rispetto ad altri modelli più leggeri, tuttavia questo non ha rappresentato un problema significativo, poiché la generazione dei contenuti è gestita centralmente nel *Back-end*, dove la latenza può essere gestita in modo più efficiente e non impatta l'esperienza utente finale in modo negativo.

La Figura 2.2 riporta la tabella comparativa contenente l'aggiornamento di *Claude 3.5 Sonnet* rispetto ad altri modelli LLM.

	Claude 3.5 Sonnet (new)	Claude 3.5 Haiku	Claude 3.5 Sonnet	GPT-4o*	GPT-4o mini*	Gemini 1.5 Pro	Gemini 1.5 Flash
Graduate level reasoning *GPQA (Diamond)*	65.0% 0-shot CoT	41.6% 0-shot CoT	59.4% 0-shot CoT	53.6% 0-shot CoT	40.2% 0-shot CoT	59.1% 0-shot CoT	51.0% 0-shot CoT
Undergraduate level knowledge *MMLU Pro*	78.0% 0-shot CoT	65.0% 0-shot CoT	75.1% 0-shot CoT	—	—	75.8% 0-shot CoT	67.3% 0-shot CoT
Code *HumanEval*	93.7% 0-shot	88.1% 0-shot	92.0% 0-shot	90.2% 0-shot	87.2% 0-shot	—	—
Math problem-solving *MATH*	78.3% 0-shot CoT	69.2% 0-shot CoT	71.1% 0-shot CoT	76.6% 0-shot CoT	70.2% 0-shot CoT	86.5% 4-shot CoT	77.9% 4-shot CoT
High school math competition *AIME 2024*	16.0% 0-shot CoT	5.3% 0-shot CoT	9.6% 0-shot CoT	9.3% 0-shot CoT	—	—	—
Visual Q/A *MMMU*	70.4% 0-shot CoT	—	68.3% 0-shot CoT	69.1% 0-shot CoT	59.4% 0-shot CoT	65.9% 0-shot CoT	62.3% 0-shot CoT
Agentic coding *SWE-bench Verified*	49.0%	40.6%	33.4%	—	—	—	—
Agentic tool use *TAU-bench*	Retail 69.2% Airline 46.0%	Retail 51.0% Airline 22.8%	Retail 62.6% Airline 36.0%	—	—	—	—

Figura 2.2: Confronto tra *Sonnet* ed altri LLM - Ottobre 2024
Update of Sonnet 3.5. URL:
https://www.anthropic.com/news/3-5-models-and-computer-use

2.5.5 Prodotti attesi

Il progetto prevede la consegna di diversi prodotti finali che soddisfino i requisiti tecnici e organizzativi stabiliti.

Di seguito vengono elencati e descritti i principali prodotti attesi:

- ***Web application*:** Lo sviluppo di una *web application* completa e funzionante, che consenta agli utenti finali di interagire con le funzionalità previste dal progetto;
- ***API RESTful*:** La realizzazione di un set di *API RESTful* che fornisca i servizi *Back-end* necessari per il funzionamento della *web application*;
- **Documento di analisi progettuale:** Un documento esaustivo che descriva nel dettaglio le scelte progettuali e i servizi AWS utilizzati.
  Questo documento deve evidenziare il ruolo di ciascun servizio, come *AWS Bedrock* per la generazione di documenti e *AWS S3* per l'archiviazione nel cloud, andando a giustificare le scelte in termini di benefici per il progetto;
- **Documentazione delle API con *Swagger*:** È prevista la generazione automatica della documentazione delle *API RESTful* tramite *Swagger*.
  Tale documentazione deve includere informazioni dettagliate sulle funzionalità

disponibili, i percorsi, i metodi supportati, i formati di richiesta e risposta, e gli eventuali codici di errore.
La documentazione deve essere costantemente aggiornata per riflettere eventuali modifiche alle API;

- **Documentazione del piano di *test*:** Un documento dettagliato che descriva il piano di *test* e collaudo della piattaforma.
  Questo piano deve includere le strategie di *test* per tutte le componenti del sistema, comprese le *API RESTful*, la *web application* e i servizi *Back-end*.
  La documentazione deve includere i *test* unitari, i *test* di integrazione, i *test* di sistema e i *test* di accettazione.
  Ogni *test* dovrà essere accompagnato da un piano di esecuzione e dai criteri di accettazione per garantire la qualità e la funzionalità del sistema in ogni sua parte.

Questi prodotti rappresentano il risultato tangibile delle attività di sviluppo e progettazione, garantendo sia la funzionalità tecnica che la trasparenza documentale del progetto.

2.6 Obiettivi personali

La scelta di intraprendere questo progetto di *stage* è stata motivata da diversi fattori che ritengo molto rilevanti per la mia crescita professionale e per il mio interesse verso tematiche innovative e tecnologiche.

Di seguito sono elencate le motivazioni principali:

- **Tema del progetto:** L'*Generative AI* è un campo estremamente affascinante e in continua evoluzione, con applicazioni che spaziano in molti settori.
  L'opportunità di lavorare su un progetto che utilizza queste tecnologie rappresenta una sfida stimolante ed un'occasione unica per approfondire le mie conoscenze in un contesto pratico;

- **Prima impressione a *STAGE-IT*:** Il colloquio tenuto con i rappresentanti aziendali a *STAGE-IT*, un evento tenuto ad Aprile 2024 che permette alle aziende di presentare i propri progetti di *stage*, è risultato molto interessante.
  L'incontro con alcuni dei dipendenti aziendali mi ha fatto una buonissima impressione, si sono dimostrati molto disponibili nel rispondere alle mie domande ed ho trovato stimolante il loro entusiasmo nell'affrontare tematiche tecnologiche avanzate;

- **Utilizzo di servizi AWS:** Sono molto interessato ai servizi AWS, che considero fondamentali nel contesto lavorativo odierno.
  AWS offre una vasta gamma di servizi *cloud*, essenziali per lo sviluppo di applicazioni moderne e scalabili, e credo che acquisire esperienza con questi strumenti possa essere un vantaggio significativo per la mia carriera lavorativa.
  Essendo Zero12 un partner AWS, questo progetto mi offre l'opportunità di lavorare con questi servizi in un contesto pratico e reale;

- **Tecnologie all'avanguardia:** Rispetto ad altre aziende con cui ho avuto colloqui, questa si distingue per l'adozione di tecnologie moderne e innovative.

L'opportunità di lavorare con tecnologie all'avanguardia rappresenta una grande opportunità di apprendimento;

- **Posizione geografica:** L'azienda si trova a circa 40 minuti dalla mia residenza, in una posizione comoda e ben collegata, che rende semplice e pratico il tragitto quotidiano;
- ***Team* giovane e dinamico:** Dopo aver parlato con alcuni studenti che avevano già svolto lo *stage* in azienda, mi è stato riferito che l'ambiente di lavoro è molto positivo.
  Il *team* giovane e dinamico è altamente collaborativo e orientato alla crescita professionale, creando un contesto ideale per lo sviluppo delle competenze.

Gli obiettivi personali rappresentano le competenze e le conoscenze che mi sono imposto di acquisire durante il mio periodo di *stage*.

Questi obiettivi sono legati al miglioramento e allo sviluppo delle proprie capacità professionali, sia dal punto di vista tecnico che umano. La loro definizione aiuta a focalizzarsi su aree specifiche di crescita e a pianificare il percorso formativo in modo mirato.

Nel contesto del mio stage, gli obiettivi personali possono essere suddivisi in tre categorie principali:

- **Obiettivi tecnici**: Si concentrano sull'acquisizione di nuove competenze tecniche, come l'apprendimento di linguaggi di programmazione, l'utilizzo di *framework*, e l'esplorazione di tecnologie avanzate.
- **Obiettivi di crescita personale**: Riguardano lo sviluppo delle *soft skills*, come la gestione del tempo, la comunicazione professionale e la comprensione del contesto aziendale.
- **Obiettivi di autonomia e collaborazione**: Focalizzati sul miglioramento della capacità di lavorare autonomamente e di collaborare efficacemente con i colleghi.

All'interno della seguente tabella sono riportati gli obiettivi personali, aventi la seguente nomenclatura:

OP-N

Dove:

- OP sta per Obiettivo Personale;
- **N** $\in N$ rappresentante il numero progressivo dell'obiettivo.

Codice Obiettivo	Descrizione Obiettivo
Obiettivi Personali	
OP-1	**Padroneggiare nuovi linguaggi e *framework*:** Acquisire competenze avanzate nello sviluppo di applicazioni utilizzando *React* per il *Front-end* e *NestJS* per il *Back-end*, implementando progetti reali che sfruttano queste tecnologie.
OP-2	**Esplorare i servizi *cloud* di AWS:** Imparare ad utilizzare servizi come *AWS Amplify*, *AWS Cognito*, *AWS S3*, e *AWS Bedrock*, comprendendo come integrarli in un'architettura scalabile e moderna.
OP-3	**Competenze in *Generative AI*:** Sviluppare un solido *know-how* nell'utilizzo di tecnologie di *Generative AI*, come l'integrazione di modelli *AI (Claude, GPT)* in progetti pratici.
Obiettivi di Crescita Personale	
OP-4	**Comprendere il settore professionale:** Ottenere una visione del contesto aziendale del settore tecnologico, analizzando flussi di lavoro e *trend* di mercato, per orientare al meglio il percorso professionale futuro.
OP-5	**Ottimizzare la gestione del tempo:** Sviluppare un approccio strutturato al lavoro, utilizzando strumenti di produttività e tecniche di prioritizzazione per rispettare scadenze e migliorare l'efficienza personale.
OP-6	**Comunicazione professionale efficace:** Rafforzare le capacità di comunicazione scritta e orale per facilitare il dialogo e le collaborazioni.
Obiettivi di Autonomia e Collaborazione	
OP-7	**Lavoro indipendente:** Incrementare la capacità di gestire *task* e progetti in modo autonomo, prendendo decisioni informate e risolvendo problemi complessi senza supervisione diretta.
OP-8	**Collaborazione proattiva:** Contribuire attivamente al lavoro di squadra, partecipando a *meeting*, condividendo idee e accogliendo *feedback* per migliorare continuamente le proprie performance.

Tabella 2.3: Obiettivi personali dello *stage*

Capitolo 3

Svolgimento dello *stage*

3.1 Pianificazione delle attività

Ho effettuato la pianificazione delle attività seguendo un approccio iterativo e collaborativo, con l'obiettivo di garantire una gestione efficace del progetto.
Durante la prima settimana, ho lavorato alla definizione delle *User Stories*, che hanno permesso di identificare e organizzare le funzionalità principali del progetto in base alla priorità e al valore per l'utente.
Per la lista delle *User Stories* si veda la sezione §3.2.1.

Successivamente, insieme al *tutor* aziendale, le attività sono state suddivise in *sprint* settimanali.
In ogni *sprint* abbiamo selezionato specifiche *User Stories* da completare, tenendo conto delle loro priorità, mantenendo un carico di lavoro sostenibile per il periodo di una settimana.
Per ciascun *sprint* ho definito dei criteri di accettazione, ovvero parametri chiari e verificabili che indicano quando un'attività può essere considerata completata.
Questi criteri, stabiliti in collaborazione con il *tutor* aziendale, hanno assicurato una qualità elevata del lavoro svolto ed una comprensione condivisa degli obiettivi.
Questo approccio mi ha permesso di monitorare costantemente i progressi e garantire un miglioramento continuo durante ogni iterazione del progetto.

3.1.1 Pianificazione settimanale

Sprint 1: Studio tecnologie (25 Settembre - 1 Ottobre)

Obiettivo: Studio delle tecnologie fondamentali per il progetto, creazione *User Stories* e della pianificazione.

Descrizione:
In questo *sprint*, l'obiettivo principale è acquisire familiarità con le tecnologie che verranno utilizzate nel progetto, come *AWS Bedrock*, *AWS Cognito*, *React*, *NestJS*, *MongoDB*, *Puppeteer* ed altri strumenti correlati.
Ho inoltre stilato le *User Stories* e la pianificazione settimanale.

Sprint 2: Implementazione autenticazione con *AWS Cognito* (2 Ottobre - 8 Ottobre)

Obiettivo: Implementare il sistema di autenticazione utilizzando *AWS Cognito*.

Descrizione:
Questo *sprint* si concentra sull'implementazione del flusso di *login* per gli utenti utilizzando *AWS Cognito*.
Ho sviluppato il sistema di autenticazione e autorizzazione, consentendo agli utenti di accedere e gestire il proprio *account*.
Ho inoltre implementato messaggi di errore chiari in caso di credenziali errate.

Sprint 3: Visualizzazione, ricerca ed eliminazione dei progetti (9 Ottobre - 15 Ottobre)

Obiettivo: Implementare la visualizzazione dei progetti e la loro ricerca.

Descrizione:
In questo *sprint*, ho sviluppato la funzionalità per visualizzare la lista dei progetti, cercare progetti specifici tramite l'applicazione di filtri e l'eliminazione di un progetto.

Sprint 4: Implementazione *preset* e bozze di progetto (16 Ottobre - 22 Ottobre)

Obiettivo: Implementare la pagina di creazione dei progetti, da cui è possibile compilare i *preset* o salvarli come bozza.

Descrizione:
In questo *sprint*, ho sviluppato la pagina per la compilazione dei *preset*, da cui è inoltre possibile salvare una bozza compilata (in parte) per essere utilizzata in un secondo momento.

Ho inoltre sviluppato la pagina che contiene le informazioni di un singolo progetto, come nome, data di creazione e tutti i capitoli che lo compongono.

Sprint 5: Generazione progetti e *download* PDF (23 Ottobre - 29 Ottobre)

Obiettivo: Implementare la generazione di progetti ed il *download* dei documenti in formato PDF.

Descrizione:
Questo *sprint* si concentra sulla generazione dei progetti utilizzando i *preset* e sulla possibilità di scaricare il progetto finale in formato PDF.
L'utente può scaricare il documento completo per l'archiviazione e/o la condivisione.

Sprint 6: Rigenerazione progetti e capitoli (30 Ottobre - 5 Novembre)

Obiettivo: Implementare la rigenerazione dei progetti e la possibilità di rigenerare capitoli specifici.

Descrizione:
In questo *sprint*, ho implementato la funzionalità per rigenerare un progetto completo o singoli capitoli, consentendo all'utente di modificare e aggiornare solo le parti necessarie del progetto, mantenendo intatti gli altri capitoli.

Sprint 7: Gestione della versione e rifiniture finali (6 Novembre - 12 Novembre)

Obiettivo: Gestire la versione dei progetti ed effettuare le rifiniture finali prima dell'attività di *testing*.

Descrizione:
In questo *sprint*, ho sviluppato la funzionalità per gestire la versione dei progetti, permettendo agli utenti di visualizzare, confrontare e ripristinare versioni precedenti. Inoltre, ho completato le ultime rifiniture del sistema, inclusi eventuali miglioramenti all'interfaccia o risoluzioni di *bug*.

Sprint 8: *Testing* e *deployment* (13 Novembre - 19 Novembre)

Obiettivo: Eseguire i *test* finali, risolvere i *bug* ed effettuare il *deploy* del sistema in produzione.

Descrizione:
Questo *sprint* si concentra sull'attività finale di *testing* del sistema.
Ho eseguito i *test* unitari, di integrazione e di performance. Una volta superati i *test*, il sistema sarà pronto per il rilascio in produzione.
Ho inoltre creato gli *Swagger* (*OpenAPI*) di tutte le API.

La Figura 3.1 riporta il diagramma di *Gantt* che rappresenta la pianificazione delle attività svolte durante il periodo di stage, suddivise per *sprint* settimanali.

Figura 3.1: Diagramma di *Gantt* delle attività svolte durante il periodo di stage

3.2 Analisi dei requisiti

L'analisi dei requisiti rappresenta un'attività cruciale in qualsiasi progetto *software*, poiché consente di definire in modo dettagliato e strutturato ciò che il sistema dovrà realizzare per soddisfare le necessità degli utenti e degli *stakeholder*.
In questo progetto, ho condotto l'analisi seguendo un approccio metodico che si è articolato in diverse fasi.

In primo luogo, ho effettuato una mappatura delle *User Stories*, un processo che ha permesso di identificare le funzionalità principali richieste dagli utenti attraverso una rappresentazione visuale e iterativa.
Successivamente, ho individuato i casi d'uso, descrivendo in dettaglio gli scenari di interazione tra gli attori del sistema e le funzionalità del *software*.
Infine, sulla base di queste informazioni, ho creato i *requisiti*, formalizzando in modo chiaro e specifico ciò che il sistema deve implementare, suddividendo i requisiti in funzionali, non funzionali e tecnologici per garantire una visione completa del progetto.

3.2.1 *User story mapping*

Introduzione

Le *User Stories* rappresentano una tecnica fondamentale nel contesto dello sviluppo *Agile*, in particolare all'interno del *framework Scrum*.
Questa metodologia mi ha consentito di creare una visione d'insieme delle funzionalità richieste dal sistema, organizzandole in modo strutturato e collaborativo[1].

Utilizzo nel contesto *scrum*

Nel contesto di *Scrum*, le *User Stories* sono utilizzate per:

- **Comprendere le esigenze degli utenti:** La mappatura aiuta a identificare le principali funzionalità del prodotto dal punto di vista degli utenti finali;
- **Definire un *backlog* strutturato:** La tecnica consente di creare un *Product Backlog* organizzato, ordinando le *User Stories* per priorità e obiettivi di rilascio;
- **Pianificare gli *sprint*:** La mappa aiuta il *Product Owner* ed il *team* ad individuare le storie da completare in ciascun *sprint*, bilanciando valore e complessità;
- **Favorire la collaborazione:** La mappa è creata in un incontro collaborativo che coinvolge il *Product Owner*, il *team* di sviluppo e, quando possibile, gli *stakeholder*, per garantire un allineamento sulle priorità.

[1] *User Stories Mapping.* URL: https://jpattonassociates.com/story-mapping/.

Struttura delle *user stories*

Ogni *user story* segue una struttura standard che include diversi elementi essenziali:

- **Priorità**, ho classificato ogni storia in base alla sua priorità, indicata con tre livelli derivati dal metodo *MoSCoW*[2]:
  - **M:** *Must have* (essenziale): funzionalità indispensabili per il sistema;
  - **S:** *Should have* (importante): funzionalità utili ma non critiche;
  - **C:** *Could have* (opzionale): funzionalità che aggiungono valore ma possono essere escluse in caso di limitazioni temporali o di risorse.
- **Descrizione**, ho descritto ogni storia in modo sintetico attraverso la notazione standard:

  Come [*utente*], voglio [*azione*], affinché [*obiettivo*].

  Ad esempio: "Come cliente, voglio visualizzare il catalogo prodotti, affinché possa scegliere cosa acquistare."

- **Criteri di accettazione**, ho definito le condizioni che devono essere soddisfatte affinché le *User Stories* possano essere considerate completate.
  Sono specifiche misurabili che servono come riferimento per i *test* e l'approvazione.

  Ad esempio, per la storia sopra, i criteri di accettazione potrebbero essere:

  - Il catalogo deve mostrare almeno 20 prodotti per pagina;
  - Ogni prodotto deve avere nome, immagine e prezzo visibili;
  - L'utente deve poter filtrare i prodotti per categoria.

[2] *MoSCoW Method*. URL: https://en.wikipedia.org/wiki/MoSCoW_method/.

User stories identificate

Di seguito le *User Stories* più significative che ho individuato durante l'analisi dei requisiti:

M - Generazione progetto tramite *preset*:

Come utente voglio poter generare un progetto con l'aiuto di *preset* predefiniti e ottimizzati, così da creare rapidamente un documento utile e strutturato senza partire da zero.

Criteri di accettazione:

1. All'interno della pagina di creazione di un progetto l'utente deve poter selezionare un *preset* predefinito dalla lista;
2. Dopo la selezione e la compilazione del *preset* il progetto deve essere generato con i dati inseriti;
3. Se la generazione avviene con successo, l'utente viene reindirizzato alla pagina di visualizzazione del singolo progetto;
4. L'utente riceve una notifica di successo.

M - Download documento progetto:

Come utente voglio poter scaricare il progetto generato in formato PDF così da poterlo condividere facilmente con il mio *team* o archiviare per un utilizzo futuro.

Criteri di accettazione:

1. Dalla pagina di visualizzazione singolo progetto l'utente deve poter scaricare il progetto in formato PDF;
2. Il *file* generato deve includere tutti i capitoli del progetto.

M - Rigenerazione di un progetto:

Come utente voglio poter rigenerare un progetto, aggiungendo o modificando informazioni, così da ottenere diverse versioni del documento per meglio adattarsi alle mie esigenze.

Criteri di accettazione:

1. L'utente deve poter accedere alla funzione di rigenerazione da un progetto esistente;
2. Deve essere possibile inserire un *Prompt* contenente le direttive da usare per la rigenerazione del progetto;
3. La versione rigenerata deve essere salvata come nuova versione del progetto.

3.2.2 Casi d'uso

I diagrammi dei casi d'uso consentono di descrivere in modo chiaro e strutturato le interazioni tra gli attori e il sistema.

Per la stesura dei casi d'uso, ho adottato una convenzione specifica, organizzata come segue:

UC[Numero] - Nome del caso d'uso

Dove:

- **UC**: acronimo di *Use Case*;
- **Numero**: identificatore progressivo del caso d'uso.
  I sottocasi sono rappresentati nella forma [Numero].[SottoNumero].

Ogni caso d'uso è corredato da una descrizione dettagliata, dalla lista degli attori coinvolti, dalla sua descrizione, dalle precondizioni e dalle postcondizioni necessarie affinché il caso d'uso possa essere eseguito correttamente.

Ho incluso esclusivamente i casi d'uso principali, omettendo quelli banali o secondari per privilegiare chiarezza e rilevanza.

Attori

Nel contesto dei casi d'uso, un attore è colui che interagisce con il sistema per ottenere un obiettivo specifico.
La Figura 3.2 mostra uno schema contenente tutti gli attori del sistema, i dettagli di ciascun attore sono descritti di seguito:

Utente non autenticato:

- **Descrizione:** L'utente non autenticato è una persona che sta cercando di accedere al sistema, ma non ha effettuato il *login*.
  Non ha accesso ai dati sensibili o alle funzionalità avanzate riservate agli utenti autenticati;
- **Ruolo:** Questo attore ha il compito di interagire con il sistema senza diritti di accesso completi.
  Il suo obiettivo principale è autenticarsi per diventare un utente con pieno accesso al sistema;
- **Attività principali:**
  - Effettuare il *login* con le proprie credenziali;
  - Visualizzare i messaggi di errore in caso di *login* fallito.

Utente autenticato:

- **Descrizione:** L'utente autenticato è una persona che ha effettuato il *login* con successo ed ha accesso completo alle funzionalità del sistema.
  Può cercare, filtrare, visualizzare, generare e rigenerare i progetti;

- **Ruolo:** Questo attore può eseguire operazioni avanzate, come la generazione di progetti, il salvataggio delle bozze ed il *download* dei progetti in formato PDF;
- **Attività principali:**
  - Visualizzare e cercare progetti;
  - Generare e rigenerare progetti;
  - Salvare e visualizzare bozze;
  - Visualizzare e scaricare progetti in formato PDF.

Large language models:

- **Descrizione:** L'attore LLM è un sistema automatizzato di generazione dei contenuti, che utilizza algoritmi avanzati di *Articial Intelligence* per generare e rigenerare progetti o singoli capitoli di progetti.
  Il LLM è responsabile dell'elaborazione delle richieste degli utenti e della generazione automatica del contenuto tecnico dei progetti;
- **Ruolo:** Il LLM è un componente del sistema che esegue la generazione di contenuti complessi in modo autonomo.
  La sua funzione principale è rispondere alle richieste dell'utente riguardo la creazione e la rigenerazione di progetti;
- **Attività principali:**
  - Ricevere richieste di generazione di progetti da parte degli utenti;
  - Generare contenuti basati su preset predefiniti;
  - Rigenerare progetti o singoli capitoli in risposta a richieste di aggiornamento;
  - Restituire i contenuti generati all'utente.

Figura 3.2: Attori dei casi d'uso

Casi d'uso individuati

UC11 - Generazione di un progetto

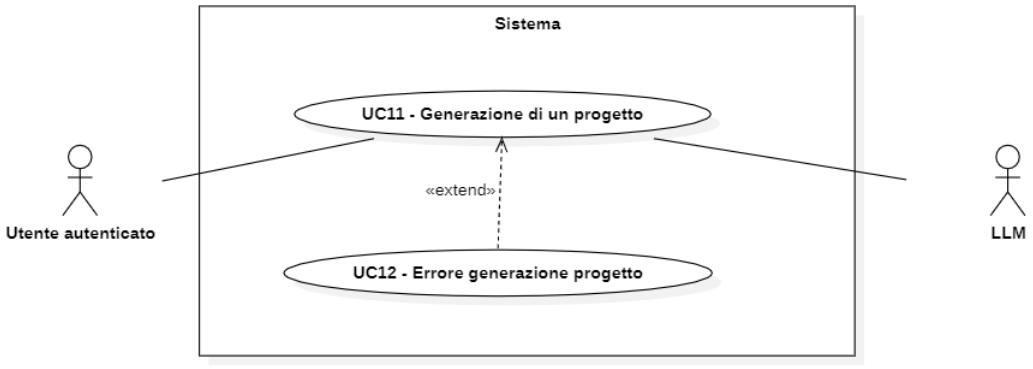


Figura 3.3: UC11 - Generazione di un progetto

- **Attori**: Utente autenticato, LLM;
- **Descrizione**: L'utente avvia la generazione di un progetto utilizzando un *preset*;
- **Precondizioni**:
  - L'utente è autenticato;
  - L'utente si trova nella pagina di creazione di un progetto ed ha selezionato un *preset*.
- **Postcondizioni**: Il progetto viene generato e l'utente vede le informazioni generate;
- **Flusso principale**:
  1. L'utente compila nella sua interezza il *preset* selezionato;
  2. L'utente richiede la generazione del progetto;
  3. Il sistema invia la richiesta al LLM;
  4. Il LLM elabora la richiesta e genera il progetto;
  5. L'utente viene reindirizzato alla pagina di visualizzazione del progetto.
- **Estensioni**: UC12 - Visualizzazione errore generazione progetto

UC13 - Rigenerazione completa di un progetto

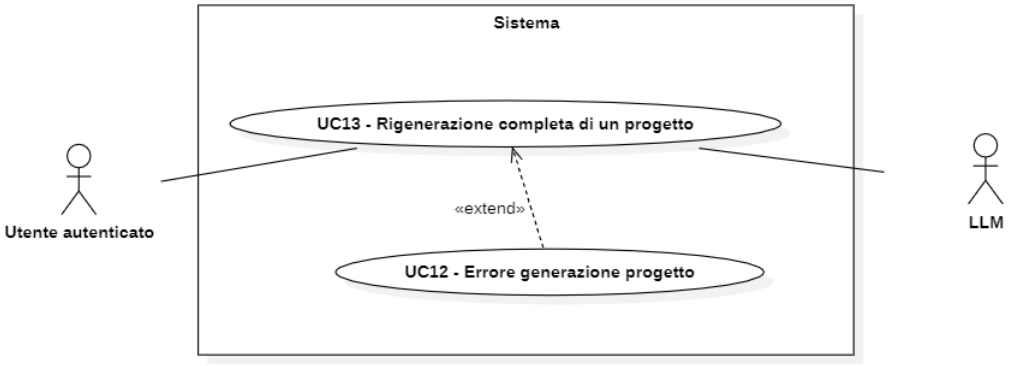


Figura 3.4: UC13 - Rigenerazione completa di un progetto

- **Attori**: Utente autenticato, LLM;
- **Descrizione**: L'utente rigenera un intero progetto, sostituendo completamente il contenuto esistente;
- **Precondizioni**:
  - L'utente è autenticato;
  - L'utente si trova nella pagina di visualizzazione del singolo progetto.
- **Postcondizioni**: Il progetto viene completamente rigenerato;
- **Flusso principale**:
  1. L'utente seleziona il pulsante di rigenerazione completa del progetto;
  2. Il sistema richiede all'utente l'inserimento di un *Prompt* su cui si baserà la rigenerazione;
  3. L'utente richiede la rigenerazione del progetto;
  4. Il sistema invia la richiesta di rigenerazione al LLM;
  5. Il LLM rigenera completamente il progetto;
  6. Il progetto rigenerato viene visualizzato all'utente.
- **Estensioni**: UC12 - Visualizzazione errore generazione progetto

UC17 - *Download* del progetto in formato PDF

- **Attori**: Utente autenticato;
- **Descrizione**: L'utente scarica il progetto in formato PDF;
- **Precondizioni**: L'utente si trova nella pagina di visualizzazione del singolo progetto ed ha selezionato il pulsante di visualizzazione del PDF;
- **Postcondizioni**: Il progetto in formato PDF viene scaricato nel dispositivo dell'utente;
- **Flusso principale**:
  1. L'utente seleziona il pulsante di *download* del progetto in formato PDF;
  2. L'utente scarica il progetto in formato PDF.

3.2.3 Tracciamento dei requisiti

Il tracciamento dei requisiti è un processo fondamentale per garantire che tutte le funzionalità e le caratteristiche desiderate del sistema siano correttamente identificate e documentate.
Per questo progetto, ho derivato i requisiti da due fonti principali: i casi d'uso e le fonti interne al progetto, come documenti tecnici, normative aziendali e linee guida.

Ho catalogato e numerato i requisiti funzionali e non funzionali per garantire una chiara tracciabilità durante lo sviluppo del sistema.
Ho inoltre associato ad ogni requisito una priorità che indica l'importanza relativa rispetto alle esigenze del progetto.

La nomenclatura dei requisiti segue il formato descritto di seguito:

R[Tipologia][Priorità]-[Numero]

Dove:

- **R**: Acronimo di *Requisito*;
- **Tipologia**: La tipologia del requisito è definita da una delle seguenti lettere:
  - **F** (Funzionale): Requisiti relativi alle funzionalità che il sistema deve fornire;
  - **NF** (Non Funzionale): Requisiti che descrivono le qualità del sistema, come le prestazioni, la sicurezza, la scalabilità, ecc.
- **Priorità**: La priorità del requisito è indicata con una delle seguenti lettere:
  - **O** (Obbligatorio): Il requisito è essenziale per il successo del progetto e deve essere soddisfatto senza eccezioni;
  - **F** (Facoltativo): Il requisito è desiderato, ma non è essenziale per il funzionamento del sistema;
  - **D** (Desiderabile): Il requisito è preferibile ma può essere rinviato o rimosso senza compromettere gravemente il progetto.

- **Numero**: Un numero intero progressivo che identifica in modo univoco ogni requisito.

In tutto ho individuato 19 requisiti funzionali e 3 requisiti non funzionali.
Di seguito mostro i più significativi.

Codice Requisito	Descrizione	Fonte
Requisiti Funzionali		
RFO-12	Il sistema deve consentire all'utente di generare un progetto utilizzando un *preset* compilato nella sua interezza	UC11
RFO-13	Il sistema deve consentire all'utente di rigenerare un intero progetto, sostituendo completamente il contenuto esistente	UC12
Requisiti Non Funzionali		
RNFO-1	Il sistema deve garantire che i tempi di risposta per la generazione o la rigenerazione di un progetto non superino i 30 secondi in condizioni di carico normale	Interna

Tabella 3.1: Tracciamento dei requisiti

3.3 Progettazione

3.3.1 Architettura del sistema

Per il progetto ho adottato un'architettura a livelli, nota come *Three-Tier Architecture*, questa architettura è composta da tre livelli distinti:

- ***Presentation Tier***: rappresenta il livello di interfaccia utente (*User Interface* (UI)) del sistema, che si occupa di ricevere *input* dall'utente e di visualizzare i risultati forniti dal livello logico;
- ***Logic Tier***: o livello logico, è il cuore del sistema, che si occupa di elaborare i dati, applicare la logica di *business* e gestire le richieste provenienti dal *Presentation Tier*;
- ***Data Tier***: gestisce la persistenza dei dati del sistema, garantendo la loro memorizzazione, recupero e integrità.

Motivazione della scelta

Ho scelto di adottare un'architettura a livelli in quanto essa rappresenta una soluzione versatile e strutturata, ideale per gestire la complessità dei sistemi moderni.
Questa scelta favorisce un'organizzazione chiara e coerente, permettendo di ottimizzare lo sviluppo e la gestione del progetto nel tempo.
Di seguito, analizzo i principali vantaggi e svantaggi di questa architettura, evidenziandone l'impatto sull'efficienza e sulla scalabilità del sistema.

Vantaggi

- **Modularità**: Ogni livello è indipendente e può essere sviluppato, testato e manutenuto separatamente;
- **Scalabilità**: È possibile scalare ogni livello in modo indipendente, ottimizzando le risorse in base alle esigenze;
- **Manutenibilità**: La separazione delle responsabilità semplifica la risoluzione dei problemi e l'implementazione di nuove funzionalità;
- **Riutilizzabilità**: I moduli di ogni livello possono essere riutilizzati in altri progetti con minime modifiche.

Svantaggi

- **Maggiore complessità**: L'implementazione e il coordinamento di più livelli richiedono un maggiore sforzo di progettazione;
- ***Overhead* di comunicazione**: La comunicazione tra i livelli può introdurre un'inevitabile latenza;
- **Dipendenze tecnologiche**: Ogni livello richiede competenze specifiche nelle tecnologie utilizzate.

Comunicazione tra i livelli

Ho implementato la comunicazione tra i livelli tramite protocolli standard e strumenti specifici:

- Tra il *Presentation Tier* e il *Logic Tier*, la comunicazione avviene tramite *API RESTful*, che permettono lo scambio di dati in formato *JSON*;
- Tra il *Logic Tier* e il *Data Tier*, è stato utilizzato l'ODM *Mongoose*, che facilita l'interazione con *MongoDB*, permettendo di definire e gestire modelli di dati in modo efficiente.

Architettura *Front-end*

L'architettura di *ReactJS* si basa sulla creazione di interfacce utente modulari e riutilizzabili, utilizzando il concetto di *components*.
Un'applicazione sviluppata con *ReactJS* segue un'architettura dichiarativa, in cui lo stato e il comportamento dell'interfaccia utente sono gestiti attraverso i *components* e gli *hooks*, garantendo flessibilità e manutenibilità del codice.

Components

I *components* sono i mattoni fondamentali di un'applicazione *ReactJS*.
Ogni *component* rappresenta una porzione dell'interfaccia utente ed è definito come una funzione od una classe in *JavaScript*.
Questi componenti possono essere combinati per creare strutture più complesse.
Ogni *componente* può gestire il proprio stato locale (*state*) e ricevere dati attraverso le *props* passate dal *parent component*.

Hooks

Gli *hooks* sono una funzionalità introdotta in *ReactJS* per gestire lo stato e gli effetti collaterali nei *functional components*, eliminando la necessità di utilizzare le classi.

Gli *hooks* che ho utilizzato includono:

- ***useState***: Permette di gestire lo stato locale di un *componente*;
- ***useEffect***: Consente di gestire gli effetti collaterali, come chiamate a API o l'aggiornamento del *DOM*;
- ***useContext***: Fornisce un modo per condividere dati tra i *componenti* senza dover passare le *props* manualmente attraverso tutti i livelli della gerarchia.

Design pattern Front-end

ReactJS supporta diversi *design pattern* che contribuiscono a migliorare l'organizzazione e la scalabilità delle applicazioni.
Quelli che ho utilizzato sono:

- ***Presentational and Container Components***: Divide i *componenti* in *presentational components*, responsabili esclusivamente della visualizzazione, e *container components*, responsabili della logica applicativa e della gestione dello stato;
- ***Custom Hooks***: Permettono di estrarre e riutilizzare logica complessa basata sugli *hooks* in funzioni personalizzate.

Architettura *Back-end*

NestJS è un *framework* per lo sviluppo di applicazioni *Back-end* che segue una struttura modulare ispirata ai principi di *Angular*.
Questo approccio facilita la creazione di applicazioni scalabili, manutenibili e testabili.

La principale caratteristica architetturale di *NestJS* è la sua organizzazione in moduli. La struttura di base di un'applicazione *NestJS* è costituita da tre componenti principali: *module*, *controller* e *service*.

Modules

I *module* sono un insieme di componenti (*controller*, *service*, *provider*) che sono raggruppati insieme per gestire una parte specifica dell'applicazione.
Un modulo in *NestJS* consente di organizzare l'applicazione in sezioni logicamente separate, facilitando la gestione del codice e la sua estendibilità.

Controllers

I *controller* sono responsabili della gestione delle richieste HTTP.
In particolare, si occupano di ricevere le richieste in entrata, chiamare la logica di *business* appropriata e restituire la risposta al *client*.
I *controller* in *NestJS* sono decorati con annotazioni specifiche, come `@Get`, `@Post`, per associare i metodi ai rispettivi *endpoint*.
Ogni *controller* è specifico per una risorsa dell'applicazione e funge da punto di ingresso per le richieste.

Services

I *service* contengono la logica di *business* dell'applicazione.
Si occupano di interagire con i *database* o con delle API esterne.
I *service* sono utilizzati dai *controller* per implementare il comportamento desiderato.
Questi sono decorati con l'annotazione `@Injectable()` per permettere la loro iniezione tramite il sistema di *Dependency Injection* (DI) di *NestJS*.

La *Dependency Injection* è un pattern di progettazione che consente di iniettare le dipendenze di un componente dall'esterno, anziché farle gestire direttamente dal componente stesso.
In *NestJS*, la DI viene utilizzata per iniettare i *service* nei *controller* e viceversa, semplificando la gestione delle dipendenze e migliorando la testabilità del codice.
Questo processo avviene automaticamente tramite il sistema di DI integrato nel *framework*.

Design pattern Back-end

NestJS si ispira a diversi *design pattern* che migliorano la modularità, la testabilità e la manutenibilità dell'applicazione.

I due *pattern* che ho utilizzato sono:

- ***Singleton Pattern***: Garantisce che una classe abbia una sola istanza e fornisce un punto di accesso globale a questa istanza.
  In *NestJS*, questo pattern è utilizzato all'interno dei *services*; grazie al sistema di *Dependency Injection* infatti, questi vengono istanziati una sola volta e condivisi in tutta l'applicazione.
  Questo approccio permette di ridurre il carico sulle risorse e assicura che i *service* mantengano uno stato coerente durante l'esecuzione dell'applicazione;
- ***Data Transfer Object* (DTO) *Pattern***: Viene usato per trasferire dati tra sistemi o livelli di un'applicazione.
  In *NestJS*, i DTO sono oggetti che definiscono la struttura dei dati che vengono inviati da e verso il *controller*.
  I DTO vengono utilizzati per convalidare, formattare e tipizzare i dati, migliorando la sicurezza e la consistenza dei dati scambiati tra il *client* e il *server*.

3.4 Codifica

3.4.1 Buone pratiche

Durante l'attività di sviluppo del progetto ho adottato diverse buone pratiche di codifica, tra cui:

- ***Arrow Functions***: Utilizzo consistente delle *arrow functions* per una sintassi più concisa e un *binding* più chiaro del `this`:

```
const add = (a, b) => a + b;
```

- **Convenzioni di *naming***:
  - *PascalCase* per la definizione dei *file ReactJs* e *NestJs*;
  - *camelCase* per variabili e funzioni;
  - *UPPER_ CASE* per le costanti.
- **Creazione di un *file .tsx* per ogni componente *ReactJs***:
  Questo approccio facilita i cambiamenti senza dover modificare il codice in più file. Ogni componente *ReactJs* ha il proprio *file .tsx*, migliorando la modularità e la manutenibilità del codice.
  Ad esempio, un componente '*SnackBar*' sarà definito in *SnackBar.tsx*';
- **Utilizzo di *Git Flow***: Struttura di *branching* per una gestione più efficiente del ciclo di vita del software.
  Ampiamente discussa nella Sezione 2.5.2 relativa ai vincoli organizzativi;
- ***Code Reviews***: Ogni modifica al codice deve essere sottoposta a *code review* da parte di un altro membro del *team*.
  Questo processo aiuta a individuare errori, migliorare la qualità del codice e condividere conoscenze tra i membri del *team*;

3.4.2 *Front-end*

La struttura della *repository Front-end* è la seguente:

- ***public***: Contiene le immagini o loghi utilizzati nell'applicazione;
- ***src***: Contiene il codice sorgente dell'applicazione, suddiviso in diverse cartelle:
  - ***tests***: Contiene tutti i *test* unitari creati per verificare il corretto funzionamento delle varie parti del codice;
  - ***components***: Contiene tutti i componenti creati, che sono le unità riutilizzabili dell'interfaccia utente;
  - ***hooks***: Contiene i *custom hooks* creati, che sono funzioni speciali di *React* per gestire lo stato e altri effetti collaterali;
  - ***interfaces***: Contiene le interfacce create, che definiscono i tipi di dati utilizzati nel progetto;
  - ***pages***: Contiene le pagine vere e proprie, che rappresentano le diverse viste dell'applicazione, come la pagina di *login* e la *dashboard*;
  - ***routes***: Contiene le *routes* del progetto, che definiscono la navigazione tra le diverse pagine e gestiscono l'autorizzazione;
  - ***services***: Contiene i servizi, in particolare quelli di autenticazione, che gestiscono la logica di *business* e le chiamate API;
  - ***app.tsx***: Il *file* principale dell'applicazione che definisce la struttura generale e include i componenti principali;
  - ***main.tsx***: Il *file* di ingresso dell'applicazione che avvia *React* e renderizza l'applicazione nel *DOM*;
- ***package.json***: Contiene le dipendenze del progetto;

- ***tsconfig.json***: Contiene le configurazioni del compilatore *TypeScript*.

Nel *Front-end* dell'applicazione ho sviluppato un totale di 8 componenti *React*, 3 *custom hooks* e 5 pagine, ciascuna progettata per offrire un'esperienza utente fluida e intuitiva.
Questi componenti e pagine sono stati realizzati con l'intento di rispondere alle esigenze specifiche degli utenti, garantendo al contempo una comunicazione efficiente con il *Back-end*. Nel dettaglio, alcune di queste pagine presentano funzionalità avanzate, come la gestione dinamica dei dati, la visualizzazione di informazioni in tempo reale e l'interazione con i servizi del *Back-end* per l'elaborazione dei dati.

Di seguito, fornisco una descrizione delle pagine più significative dell'applicazione, evidenziando le loro principali funzionalità e le modalità di interazione con il *Back-end*.

Pagina di creazione di un progetto

Questa pagina consente agli utenti di creare nuovi progetti fornendo tutte le informazioni necessarie o, in alternativa, di salvare una bozza per completare il lavoro successivamente.

- **Funzionalità principali**: La pagina include un *form* dinamico che richiede all'utente di selezionare un *preset* e compilarlo per poter andare a generare un progetto.
  Durante l'inserimento, viene effettuata una validazione lato *client* per assicurarsi che tutti i campi obbligatori siano compilati e che i dati forniti rispettino i formati richiesti.
  L'utente ha la possibilità di salvare il *preset* come bozza o di richiedere la generazione del progetto;

- **Interazione con il *Back-end***: Una volta completata la compilazione del *form*, i dati vengono inviati al *Back-end* tramite una chiamata API *POST* che si occupa della generazione del progetto.

Pagina di dettaglio progetto

Questa pagina fornisce una vista approfondita di un singolo progetto, mostrando tutte le informazioni associate e consentendo di accedere al documento PDF generato.

- **Funzionalità principali**: La pagina presenta dettagli completi del progetto, inclusi nome, data di creazione e tutti i capitoli che lo compongono.
  È possibile visualizzare il PDF generato direttamente all'interno di un visualizzatore integrato oppure di scaricarlo per un utilizzo futuro.
  Inoltre, la pagina permette di andare a rigenerare il progetto nella sua totalità o in parte;

- **Interazione con il *Back-end***: All'apertura della pagina, una chiamata API *GET* recupera tutte le versioni del progetto.
  Il PDF, generato precedentemente nel *Back-end*, è reso disponibile tramite un *URL* fornito da *AWS S3*, garantendo sicurezza e velocità di accesso.

La *Listing* 3.1 mostra la funzione di *retrieve* di tutte le versioni del progetto, si noti la riga `projectsData[projectsData.length - 1]`, questa va ad impostare come progetto selezionato l'ultima versione disponibile.

```
const fetchProject = async () => {
  try {
    const response = await get(`/projects/${id}`);
    if (response && response.data) {
      const projectsData = response.data as Project[];
      setProjects(projectsData);
      setSelectedProject(
        projectsData.length > 0
          ? projectsData[projectsData.length - 1]
          : null
      );
    } else {
      throw new Error("Invalid response format");
    }
    setLoading(false);
  } catch (err) {
    showSnackbar({
      message: "Failed to fetch project details.",
      severity: "error",
    });
    setLoading(false);
  }
};
```

Listing 3.1: Funzione di *retrieve* di tutte le versioni dei progetti

3.4.3 *Back-end*

La struttura della *repository* *Back-end* è la seguente:

- ***src***: Contiene varie cartelle organizzate per funzionalità, tra cui:
  - ***auth***: Contiene i *middleware* per l'autenticazione delle API;
  - ***aws-bedrock***: Contiene la logica della generazione dei progetti;
  - ***pdf***: Contiene la logica per la generazione e salvataggio su *AWS S3* dei documenti PDF;
  - ***utils***: Contiene tutti i *mock* utili per il *testing* o funzioni di supporto globale.
- ***package.json***: Contiene le dipendenze del progetto;
- ***DockerFile***: Contiene il file *Docker* per la creazione del *container* per *MongoDB*;

Per ogni cartella all'interno di *src* (ad esclusione delle cartelle ***auth*** e ***utils***), si ha una struttura comune che include:

- una cartella *dto* contenente i *Data Transfer Object* relativi alla funzionalità;
- una cartella *schemas* contenente lo schema *Object Data Modeling* relativo all'entità gestita;

- un *controller* per la gestione delle richieste HTTP;
- un *module* per l'iniezione delle dipendenze;
- un *service* per la logica di *business*;
- i *test* dei *controller* e dei *service*.

Endpoints **delle API**

Nel *Back-end* dell'applicazione, ho progettato e sviluppato un totale di 5 *endpoints*, che complessivamente implementano 11 API, ognuna delle quali svolge un ruolo fondamentale nella gestione dei dati e nell'esecuzione delle operazioni richieste dall'applicazione.

Ho implementato questi *endpoints* per garantire una comunicazione fluida e sicura tra il *Front-end*, il *Back-end* ed i servizi esterni, assicurando che ogni componente del sistema possa interagire in modo efficiente.
Inoltre, ho definito 8 DTO, che sono essenziali per la corretta gestione, validazione e trasmissione dei dati tra le diverse componenti del sistema, permettendo di mantenere l'integrità e la coerenza delle informazioni.

Di seguito, approfondisco gli *endpoints* più rilevanti, focalizzandomi sulle funzionalità avanzate che implementano e analizzando le modalità con cui questi *endpoints* interagiscono con i servizi esterni, garantendo una gestione ottimale delle risorse e una comunicazione efficiente tra le varie parti del sistema.

`/aws-bedrock`

Ho implementato l'*endpoint* `/aws-bedrock` per gestire la generazione automatica dei progetti sfruttando le capacità di *AWS Bedrock*.
La sua funzione principale è quella di orchestrare la comunicazione tra il sistema e un modello di linguaggio (LLM) avanzato, consentendo la creazione di progetti dettagliati.

Per garantire consistenza e affidabilità nel risultato, ho utilizzato la libreria *LangChain*, un *framework* che supporta la definizione di output strutturati direttamente dall'LLM, permettendo di definire uno schema di *output* prevedibile.

La *Listing* 3.2 mostra la dichiarazione della LLM e la funzione di generazione dei progetti implementata nel *service*, che si occupa di invocare il modello di linguaggio e di restituire il risultato al *Front-end*.

```
private readonly llm: ChatBedrockConverse;
  constructor() {
    // declare the LLM
    this.llm = new ChatBedrockConverse({
      model: "anthropic.claude-3-5-sonnet-20241022-v1:5",
      temperature: 0.3,
      topP: 0.9,
      maxTokens: 4096,
      region: "eu-central-1",
      // AWS credentials
      credentials: fromIni({ profile: "dev" }),
    });
  }
  async generateProject(
    prompt: string,
    promptId: string,
    projectId: string
  ): Promise<CreateProjectDTO> {

    try {
      // Define the output schema
      const structuredLlm = this.llm.withStructuredOutput(
          ProjectSchema);
      // Invoke the LLM
      const response = await structuredLlm.invoke(prompt);
      const result : CreateProjectDTO = response;

      // Assign the promptId
      result.promptId = promptId;
      // Assign the projectId
      result.projectId = projectId;

      return result;
    } catch (error) {
      throw new Error(
        `Failed to generate project with AWS Bedrock: ${error.
            message}`
      );
    }
  }
```

Listing 3.2: Dichiarazione LLM e sua invocazione

/pdf

Ho creato l'*endpoint* `/pdf` per gestire la generazione dei documenti PDF ed il loro salvataggio su *AWS S3* a partire dai progetti creati.
Questo *endpoint* svolge un ruolo fondamentale nel flusso di lavoro complessivo, poiché consente agli utenti di ottenere una rappresentazione visiva e condivisibile dei progetti generati.
Una volta creato, il PDF viene caricato su un *bucket AWS S3*, assicurandone la memorizzazione a lungo termine e l'accessibilità tramite un *URL* univoco.

La *Listing* 3.3 riporta la funzione di generazione del PDF, si noti la popolazione del template HTML con i dati del progetto e il salvataggio del PDF generato su *AWS S3*.

```
async generatePdf(user: User, project: Project): Promise<string>
    {
    // Populate the HTML template
    const content = await this.populateTemplate(project);
    // Get the browser instance
    const browser: Browser = await this.getBrowser();

    const page: Page = await browser.newPage();
    // Set the content of the page
    await page.setContent(content);

    try {
      // Generate the PDF
      const pdfBuffer: Uint8Array = await page.pdf({
        format: "A4",
        printBackground: true,
      });
      // Save the PDF on AWS S3
      this.savePdf(user, pdfBuffer, project.projectId, project.
          projectName, project.version);
      // Return the signed URL to the frontend
      return this.getSignedUrl(user, project.projectId, project.
          projectName, project.version);
    } catch (error) {
      throw new Error("Failed to generate PDF");
    } finally {
      await page.close();
      await this.closeBrowser();
    }
  }
```

Listing 3.3: Funzione di generazione del PDF, con salvataggio su *AWS S3*

3.5 Verifica

La verifica è un'attività cruciale del ciclo di vita del *software*, finalizzata a garantire che il prodotto sviluppato soddisfi i requisiti specificati e sia privo di errori o difetti. Questa attività si concentra sul controllo della correttezza, della completezza e della qualità del *software*, sia a livello di codice che di comportamento.

La verifica può essere suddivisa in due approcci principali:

- **Analisi statica**: È un processo che analizza il *software* senza eseguirlo.
  Si basa su tecniche di analisi statica, come le revisioni manuali del codice, l'uso di strumenti automatici per il controllo delle regole di codifica o per il *linting* del codice.
  Questo approccio consente di individuare errori, violazioni di standard, vulnerabilità e problemi di stile a livello di codice sorgente.
- **Analisi dinamica**: Si basa sull'esecuzione del *software* per testarne il comportamento in ambienti controllati o reali.
  Questo approccio verifica che il sistema funzioni correttamente con *input* specifici e che produca gli *output* attesi.
  Include attività come i *test* unitari, *test* di integrazione e *test* di sistema.

L'utilizzo combinato di analisi statica e dinamica è essenziale per garantire un livello elevato di qualità del *software*.

3.5.1 Analisi statica

Ho effettuato analisi statica per garantire che il codice sviluppato fosse conforme agli standard di qualità richiesti, riducendo la possibilità di errori e migliorandone la leggibilità e la manutenibilità.
Per questa attività ho utilizzato strumenti consolidati come *Prettier* ed *ESLint*, configurati per integrarsi con il flusso di lavoro del progetto.

3.5.2 Analisi dinamica

Ho effettuato analisi dinamica per verificare il corretto funzionamento delle diverse componenti del sistema, concentrandomi in particolare sul *Back-end*, considerato il cuore del progetto.
Il testing sul *Front-end* è stato affrontato durante un'attività successiva, con un approccio meno approfondito rispetto a quello dedicato al *Back-end*.

Di seguito, descrivo in dettaglio le attività di *testing* dinamico effettuate sia sul *Front-end* che sul *Back-end*.

Testing del *Back-end*

Ho sviluppato *test* unitari per tutti i *controller* e i *service*, assicurando una copertura completa delle funzionalità fondamentali del sistema.

Durante l'intero ciclo di vita del *software*, ho adottato il modello a V (illustrato in Figura 3.8), che integra il *testing* sin dalle attività iniziali dello sviluppo.
Con un approccio iterativo, per ogni *endpoint* creato ho immediatamente sviluppato i relativi *test*, garantendo così la loro correttezza.

Ho implementato un totale di 70 *test* unitari nel *Back-end*, coprendo una vasta gamma di funzionalità chiave del sistema e raggiungendo una copertura complessiva del 75%.
Focalizzandosi esclusivamente sui *services* e *controller* degli *endpoints* da me sviluppati, la copertura raggiunge il 100%.
Ho dedicato particolare attenzione alla funzionalità di generazione dei progetti tramite LLM, che rappresenta una parte cruciale del flusso di lavoro dell'applicazione.
Questa funzionalità, infatti, è essenziale per la creazione automatizzata dei progetti e per garantire l'integrità dei dati generati.

Per simulare un ambiente di *test* il più realistico possibile, ma senza dipendere direttamente dalle risorse esterne, ho creato dei *mockup* per simulare il comportamento di funzionalità critiche.
In particolare, ho sviluppato *mockup* per le interazioni con i servizi *AWS Bedrock* e *AWS S3*, utilizzati rispettivamente per la chiamata ai modelli di linguaggio e per la gestione dei documenti.
Questi *mockup* hanno permesso di replicare il comportamento atteso di questi servizi, consentendo il *testing* senza richiedere accesso diretto a tali risorse esterne.

Questo approccio ha permesso di testare e validare ogni componente sin dalla sua creazione, riducendo al minimo i rischi di errori e facilitando il *debug* nelle attività successive di sviluppo.

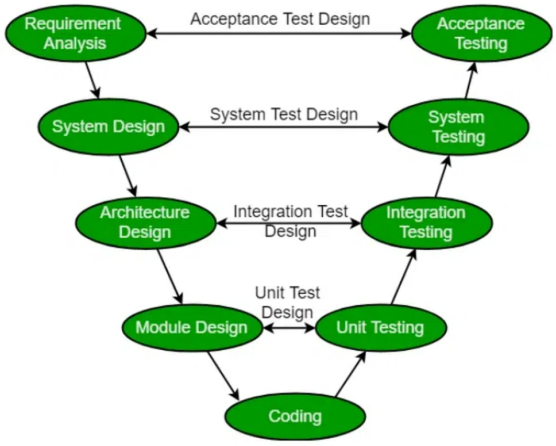


Figura 3.5: Modello a V sviluppo *software*
V-Model. URL:
https://www.geeksforgeeks.org/software-engineering-sdlc-v-model/

Testing del *Front-end*

Il *testing* del *Front-end* è stato affrontato principalmente nell'ultimo *sprint* del progetto, con un focus minore rispetto al *Back-end*, in quanto le funzionalità del *Front-end* erano strettamente dipendenti dal corretto funzionamento del *Back-end*.

Durante questo *sprint*, il mio approccio si è concentrato principalmente sul *test* delle funzioni logiche delle pagine, così come sulle chiamate API, per assicurarmi che i dati venissero correttamente recuperati e visualizzati.
Le chiamate API sono state verificate per garantire che rispondessero correttamente in termini di formati e tempistiche.

Ho implementato un totale di 32 *test* unitari per il *Front-end*, coprendo sia le logiche delle pagine che la gestione delle chiamate alle API.
La copertura complessiva del codice è pari al 50%; tuttavia, considerando esclusivamente i *components* da me sviluppati, la copertura raggiunge l'80%.
Inoltre, ho utilizzato *React Testing Library* per eseguire *test* sui singoli componenti, in modo da verificarne la corretta visualizzazione e interazione con l'utente.
Questo strumento mi ha permesso di testare i componenti in modo isolato, simulando gli eventi utente (come i *clic* e la compilazione di moduli) e controllando la corretta risposta da parte dell'interfaccia utente.

Il *focus* del *testing* del *Front-end* è stato limitato rispetto a quello del *Back-end*, ma ha comunque permesso di garantire che l'interfaccia utente rispondesse correttamente alle azioni degli utenti e che tutte le chiamate alle API si comportassero come previsto.

Test di rendimento

Nel corso dello sviluppo, ho dedicato particolare attenzione alla valutazione delle prestazioni del *Back-end*, eseguendo una serie di *test* di rendimento focalizzati sulla funzionalità di generazione dei progetti, che rappresenta una delle operazioni più critiche del sistema.

Questi *test* sono stati progettati per misurare in dettaglio i tempi di risposta del sistema durante la creazione di un progetto, al fine di garantire che le operazioni avvenissero in maniera tempestiva e senza ritardi significativi.
Ho esaminato vari scenari, inclusi i casi in cui il sistema gestisce grandi quantità di dati, per assicurarmi che fosse in grado di rispondere in modo efficiente anche in situazioni di carico moderato.

Inoltre, i *test* hanno incluso simulazioni di più richieste simultanee, per analizzare come il sistema si comportasse in scenari di stress e per verificare che non ci fossero colli di bottiglia o rallentamenti significativi durante il picco di utilizzo.

Questi *test* di rendimento sono stati fondamentali per assicurare che il *Back-end* fosse non solo funzionale, ma anche scalabile e in grado di gestire l'elaborazione di progetti in modo efficiente, riducendo i rischi di inefficienze o interruzioni delle prestazioni durante l'uso del sistema.

3.6 Validazione

La validazione è il processo che verifica se il prodotto finale soddisfa i requisiti e le aspettative degli utenti o delle parti interessate.
A differenza della verifica, che si concentra sull'accuratezza e completezza tecnica delle singole componenti (ad esempio, controllando che il codice sia conforme alle specifiche), la validazione si focalizza sull'effettivo valore e utilità del *software* nel contesto operativo previsto.

3.6.1 Test di accettazione

Ho condotto i *test* di accettazione in collaborazione con il *tutor* aziendale, con l'obiettivo di verificare che il progetto soddisfacesse i requisiti funzionali e non funzionali definiti durante l'attività di analisi.

Durante la sessione di *test*, il tutor aziendale ha valutato il comportamento del sistema rispetto alle aspettative aziendali e alle specifiche iniziali. La valutazione finale è stata positiva, confermando che il *software* è pronto per essere utilizzato nell'ambiente di produzione e soddisfa pienamente le esigenze del progetto.

3.6.2 Presentazione del progetto

L'ultima settimana di *stage* mi è stata richiesta dall'azienda una presentazione finale del progetto ai colleghi aziendali, con l'obiettivo di illustrare il lavoro svolto, i problemi riscontrati durante lo sviluppo e le possibili migliorie da apportare in futuro.

Durante la presentazione ho spiegato lo scopo del progetto e le tecnologie utilizzate, evidenziando il valore aggiunto che il progetto potrebbe apportare all'azienda, ho inoltre mostrato una *demo* del prodotto per dimostrare le sue funzionalità in tempo reale.

La presentazione è stata accolta in modo molto positivo dai partecipanti.
Al termine, sono state poste alcune domande per chiarire alcuni dettagli implementativi del progetto. In particolare, un collega ha manifestato interesse riguardo alla creazione dei *Prompt* utilizzati per la generazione dei progetti ed il loro processo di salvataggio, avviando un confronto costruttivo sull'argomento.
Questo ha confermato l'interesse e la rilevanza del lavoro svolto all'interno del contesto aziendale.

3.7 Risultati ottenuti

3.7.1 Prodotto realizzato

La Figura 3.6 mostra la pagina di *login*, che consente agli utenti di accedere inserendo le proprie credenziali.
In caso di problemi, è disponibile un'opzione per reimpostare la *password* cliccando su "*Password* dimenticata".

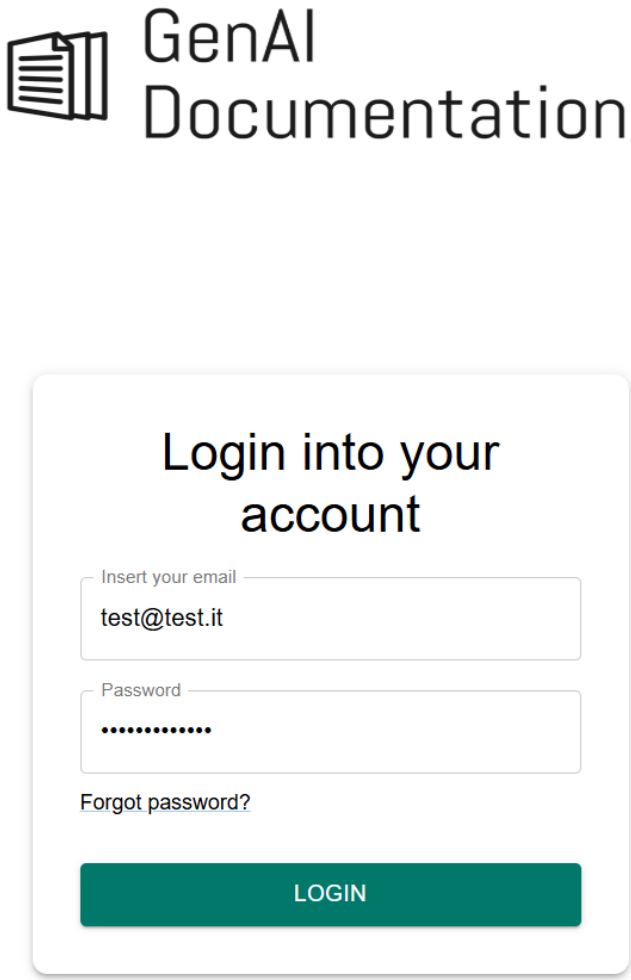


Figura 3.6: Schermata di *login* del progetto

La Figura 3.7 mostra la *dashboard*, che offre una panoramica dei progetti generati in precedenza e delle bozze di progetto, con la funzionalità, sia per progetti che per le bozze, di cercarle per nome o filtrarle per data.
Da questa pagina, gli utenti possono accedere ai dettagli di un singolo progetto o iniziare la generazione di un nuovo progetto, oltre che effettuare il *logout*.

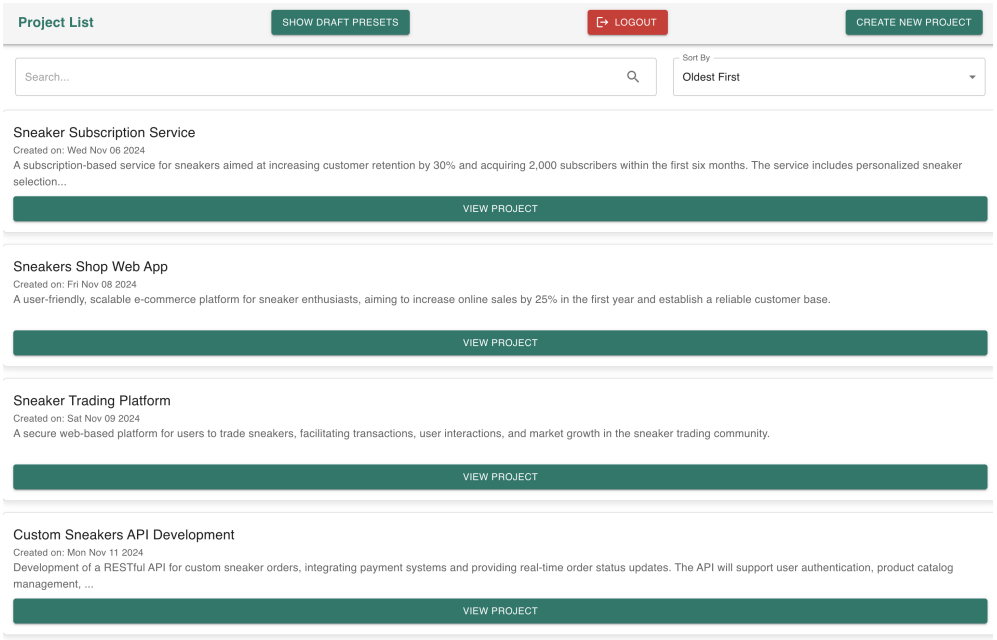


Figura 3.7: Schermata della *dashboard* del progetto

La Figura 3.8 riporta la pagina di generazione di un progetto che permette di creare nuovi progetti inserendo il nome, selezionando un *preset* e completandolo in tutte le sue sezioni.
Una volta compilato, l'utente può procedere con la generazione del progetto.
È inoltre possibile salvare il *preset* parzialmente compilato come bozza per riprendere la sua compilazione in un secondo momento.

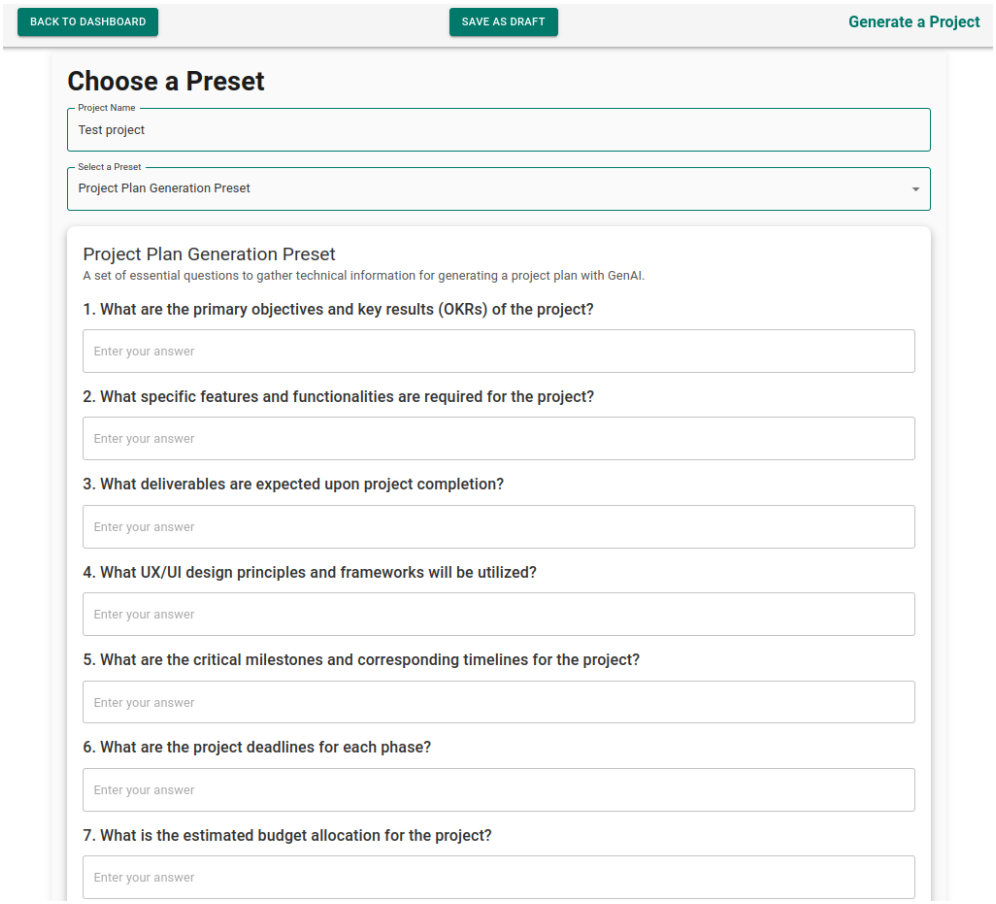


Figura 3.8: Schermata di generazione del progetto

La Figura 3.9 mostra la pagina di visualizzazione di un singolo progetto che offre una descrizione completa di tutte le informazioni relative al progetto, inclusi i dettagli tecnici ed il PDF generato, che può essere visualizzato direttamente nella pagina.

Inoltre gli utenti possono scegliere di rigenerare l'intero progetto o una singola sezione, utilizzando una finestra che consente di inserire un *Prompt* personalizzato, come riporta la Figura 3.10.

Figura 3.9: Schermata di dettaglio del progetto

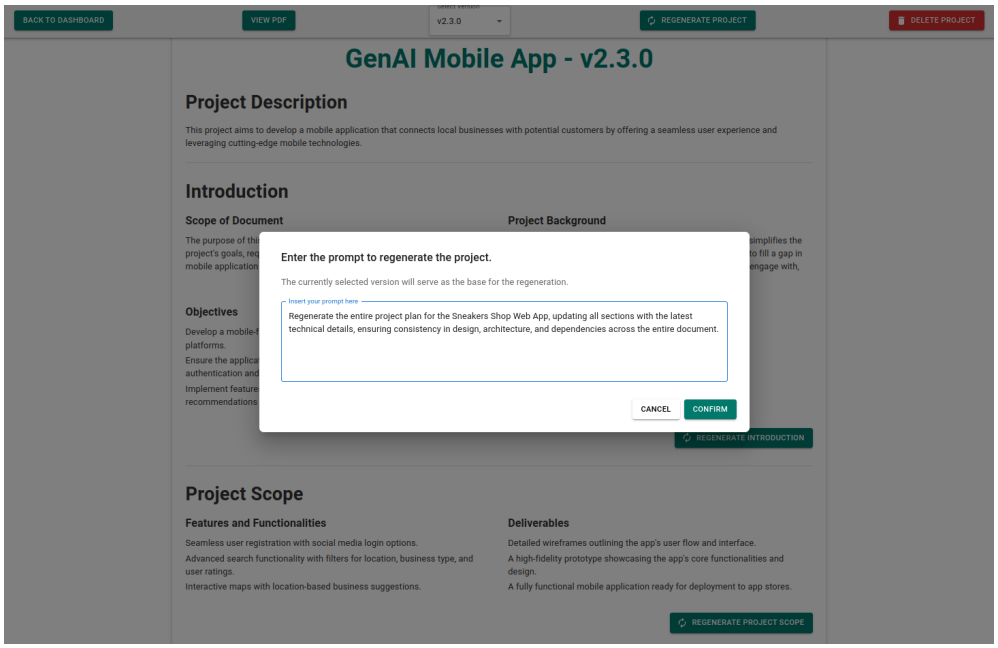


Figura 3.10: *Prompt* per rigenerare il progetto

Infine la Figura 3.11 riporta la pagina di visualizzazione del *Swagger* delle API, essa fornisce una documentazione interattiva e dettagliata delle API disponibili nel sistema. Attraverso questa pagina, gli utenti possono esplorare i vari *endpoint*, visualizzarne le descrizioni, i parametri richiesti e le risposte previste.
Inoltre, è possibile effettuare chiamate di prova direttamente dalla pagina, inserendo i parametri necessari e verificando i risultati in tempo reale.

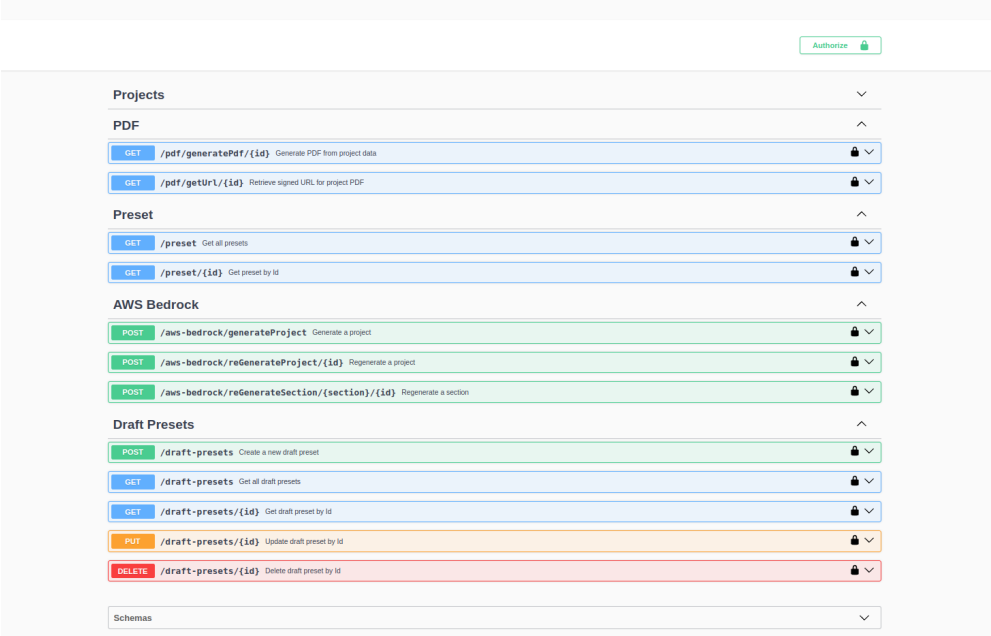


Figura 3.11: *Swagger* delle API create

3.7.2 Copertura dei requisiti

Ho soddisfatto tutti i requisiti funzionali e non funzionali, ad eccezione di un requisito funzionale desiderabile e di un requisito non funzionale, che discuto qui sotto.
Ho implementato e testato ogni funzionalità prevista, rispettando pienamente l'analisi dei requisiti iniziale.

Requisito RNFO-1: Tempi di generazione inferiori a 30 Secondi

Non ho soddisfatto il requisito RNFO-1, che prevedeva tempi di generazione dei progetti inferiori a 30 secondi, infatti l'LLM che ho scelto per la generazione dei progetti (*Claude 3.5 Sonnet*), impiega in media circa 1 minuto e 20 secondi per completare il processo.

Tuttavia, insieme al *tutor* aziendale, ho concordato che tale tempistica fosse accettabile, considerando l'elevata qualità e il livello di dettaglio forniti dal modello.
In futuro, un'ottimizzazione delle prestazioni potrebbe ridurre ulteriormente questi tempi.

3.7.3 Materiali prodotti

Metrica	Quantità
Documenti	2
Componenti *React*	8
API	11
Linee di codice	11.941
Numero di *file*	113
***Test* implementati**	102

Tabella 3.2: Tabella dei materiali prodotti

Capitolo 4

Retrospettiva finale

4.1 Raggiungimento degli obiettivi

4.1.1 Obiettivi aziendali

Riporto gli obiettivi aziendali indicati nella Sezione §2.4 e ne descrivo il grado di soddisfacimento conseguito a fine *stage*.

Codice Obiettivo	Descrizione Obiettivo	Soddisfatto
Obiettivi Obbligatori		
OO-1	**Apprendimento delle tecnologie di sviluppo:** Acquisire competenze pratiche nell'uso di *React*, *NestJS* e *MongoDB* per la progettazione e lo sviluppo di applicazioni.	Sì
OO-2	**Gestione del versionamento del codice:** Apprendere l'uso di *Git* e adottare *Git Flow* come metodologia per il controllo delle versioni e la collaborazione.	Sì
OO-3	**Analisi e scelta del LLM:** Valutare i modelli disponibili per selezionare quello più adatto al progetto.	Sì
OO-4	**Introduzione alle metodologie agili:** Familiarizzare con le metodologie agili di sviluppo per la gestione efficace di progetti.	Sì
OO-5	**Pianificazione e gestione giornaliera:** Imparare a gestire *task* e obiettivi giornalieri allineati al piano di lavoro.	Sì
OO-6	**Sviluppo di una *web app*:** Progettare e realizzare un'applicazione *web* per consentire l'interazione dell'utente con la piattaforma.	Sì
OO-7	**Sviluppo ed integrazione con *Generative AI*:** Implementare i flussi logici del progetto e integrare i servizi di *Generative AI* scelti.	Sì

Codice Obiettivo	Descrizione Obiettivo	Soddisfatto
OO-8	**Documentazione API:** Creare una documentazione *Swagger* per le API sviluppate.	Sì
OO-9	**Documento di analisi progettuale:** Redigere un documento tecnico che descriva l'architettura e le componenti principali della piattaforma.	Sì
OO-10	***User Story Mapping:*** Realizzare una mappatura delle *User Stories* per descrivere e organizzare i requisiti del progetto.	Sì
Obiettivi Desiderabili		
OD-1	**Comparazione tra modelli LLM:** Effettuare un'analisi comparativa tra almeno due LLM per verificarne le differenze in termini di prestazioni, funzionalità e costi.	No
Obiettivi Facoltativi		
OF-1	**Piattaforma di amministrazione:** Creare una piattaforma *admin* per la gestione dei *Preset* e dei modelli LLM.	No

Tabella 4.1: Obiettivi aziendali dello *stage*

Sono riuscito a soddisfare il 100% degli obiettivi obbligatori del progetto, garantendo il raggiungimento di tutte le funzionalità essenziali.
Tuttavia, non sono riuscito a completare alcuno degli obiettivi desiderabili o facoltativi, a causa di limitazioni tecniche e di tempo.

In particolare, non ho raggiunto l'obiettivo OD-1 (Comparazione tra modelli LLM) a causa della complessità tecnica nell'integrare diversi modelli con *AWS Bedrock* e nell'effettuare un'analisi approfondita delle loro prestazioni.
Inoltre, il tempo a disposizione non è stato sufficiente per completare le configurazioni necessarie.

Allo stesso modo, non sono riuscito a completare lo sviluppo della piattaforma di amministrazione (OF-1). Questo obiettivo, essendo secondario rispetto alle priorità principali del progetto, non ha potuto ricevere l'attenzione necessaria entro i tempi disponibili.

4.1.2 Obiettivi personali

Riporto gli obiettivi personali indicati nella Sezione §2.6 e ne descrivo il grado di soddisfacimento conseguito a fine *stage*.

Codice Obiettivo	Descrizione Obiettivo	Soddisfatto
Obiettivi Personali		
OP-1	**Padroneggiare nuovi linguaggi e *framework*:** Acquisire competenze avanzate nello sviluppo di applicazioni utilizzando *React* per il *Front-end* e *NestJS* per il *Back-end*, implementando progetti reali che sfruttano queste tecnologie.	Sì
OP-2	**Esplorare i servizi *cloud* di AWS:** Imparare ad utilizzare servizi come *AWS Amplify*, *AWS Cognito*, *AWS S3*, e *AWS Bedrock*, comprendendo come integrarli in un'architettura scalabile e moderna.	Sì
OP-3	**Competenze in *Generative AI*:** Sviluppare un solido *know-how* nell'utilizzo di tecnologie di *Generative AI*, come l'integrazione di modelli *AI (Claude, GPT)* in progetti pratici.	Sì
Obiettivi di Crescita Personale		
OP-4	**Comprendere il settore professionale:** Ottenere una visione del contesto aziendale del settore tecnologico, analizzando flussi di lavoro e *trend* di mercato, per orientare al meglio il percorso professionale futuro.	Sì
OP-5	**Ottimizzare la gestione del tempo:** Sviluppare un approccio strutturato al lavoro, utilizzando strumenti di produttività e tecniche di prioritizzazione per rispettare scadenze e migliorare l'efficienza personale.	Sì
OP-6	**Comunicazione professionale efficace:** Rafforzare le capacità di comunicazione scritta e orale per facilitare il dialogo e le collaborazioni.	Sì
Obiettivi di Autonomia e Collaborazione		
OP-7	**Lavoro indipendente:** Incrementare la capacità di gestire *task* e progetti in modo autonomo, prendendo decisioni informate e risolvendo problemi complessi senza supervisione diretta.	Sì
OP-8	**Collaborazione proattiva:** Contribuire attivamente al lavoro di squadra, partecipando a *meeting*, condividendo idee e accogliendo *feedback* per migliorare continuamente le proprie performance.	Parzialmente

Tabella 4.2: Obiettivi personali dello *stage*

Ho raggiunto tutti i miei obiettivi personali, ad eccezione dell'obiettivo OP-8 (Collaborazione proattiva), che sono riuscito a soddisfare solo in parte.

In particolare, il mio obiettivo di contribuire attivamente al lavoro di squadra, partecipando a *meeting*, condividendo idee e accogliendo *feedback*, non è stato pienamente raggiunto.
Anche se ho preso parte a diversi *meeting*, ho lavorato principalmente in autonomia e non mi sono state fornite opportunità di collaborare attivamente con un *team*.

4.1.3 Difficoltà incontrate

Durante lo sviluppo del progetto, ho affrontato alcune difficoltà che hanno richiesto interventi specifici per essere mitigate:

- Integrazione del servizio di autenticazione *AWS Cognito* nel *Front-end*: Ho riscontrato problematiche nella configurazione e nell'integrazione del servizio di autenticazione AWS Cognito con il *Front-end*.
  Per superare questa difficoltà, ho richiesto supporto al *team* *Front-end*, che mi ha fornito indicazioni e soluzioni utili per completare l'integrazione;
- *Output* predefinito e standardizzato dall'LLM: Un'altra difficoltà significativa è stata ottenere dall'LLM un *output* sempre predefinito e strutturato in modo coerente.
  Per risolvere questo problema, ho utilizzato *LangChain*, che mi ha permesso di definire un formato di *output* strutturato e migliorare la consistenza delle risposte generate.

4.2 Crescita professionale

4.2.1 Difficoltà incontrate

Durante lo sviluppo del progetto, ho affrontato alcune difficoltà che hanno richiesto interventi specifici per essere mitigate:

- Integrazione del servizio di autenticazione *AWS Cognito* nel *Front-end*: Ho riscontrato problematiche nella configurazione e nell'integrazione del servizio di autenticazione AWS Cognito con il *Front-end*.
  Per superare questa difficoltà, ho richiesto supporto al *team* *Front-end*, che mi ha fornito indicazioni e soluzioni utili per completare l'integrazione;
- *Output* predefinito e standardizzato dall'LLM: Un'altra difficoltà significativa è stata ottenere dall'LLM un *output* sempre predefinito e strutturato in modo coerente.
  Per risolvere questo problema, ho utilizzato *LangChain*, che mi ha permesso di definire un formato di *output* strutturato e migliorare la consistenza delle risposte generate.

4.2.2 Competenze acquisite

Nel corso del progetto ho sviluppato e perfezionato diverse competenze, che possono essere suddivise in tre categorie principali:

- **Competenze tecniche**: Ho approfondito e migliorato significativamente la mia conoscenza di *TypeScript*, applicandolo con successo in un progetto complesso nel contesto dello sviluppo *web*.
  Ho inoltre acquisito competenze nell'utilizzo di diversi servizi AWS, con un focus particolare su *AWS Bedrock*, dedicato alla *Generative AI*.
  Questo mi ha permesso di affrontare aspetti tecnici avanzati e di integrare nuove tecnologie nel progetto in maniera efficace;
- **Competenze metodologiche**: Ho avuto modo di lavorare seguendo un approccio *agile*, partecipando attivamente agli *sprint* e rispettandone le tempistiche.
  Inoltre, ho affinato la mia capacità di organizzazione autonoma, definendo con precisione le priorità e gestendo le attività necessarie per portare a termine il lavoro in modo efficiente;
- **Competenze personali**: Grazie alla partecipazione alle *sprint review* e alla presentazione aziendale del progetto, ho migliorato le mie competenze comunicative, imparando a esporre il mio lavoro in maniera chiara, strutturata e professionale.
  Ho anche sviluppato ulteriormente le mie capacità di *problem solving*, trovando soluzioni creative e pratiche alle difficoltà incontrate durante le diverse attività del progetto.

4.3 Università e mondo del lavoro

L'esperienza di *stage* mi ha permesso di comprendere in maniera più profonda come università e mondo del lavoro siano strettamente collegati, rappresentando due facce della stessa medaglia.
Da un lato, l'università offre una formazione teorica solida e strutturata, che è indispensabile per affrontare qualsiasi contesto professionale.
Dall'altro lato, è nel mondo del lavoro che queste conoscenze trovano una vera applicazione, consentendo di affinare, consolidare e trasformare le basi teoriche in competenze pratiche e operative.

In questo contesto, il corso di "Ingegneria del *software*" ha giocato un ruolo fondamentale nel mio percorso.
Durante il corso ho avuto l'opportunità di apprendere concetti cruciali per lo sviluppo e la gestione di progetti *software*, come la progettazione, la gestione dei requisiti e l'organizzazione del lavoro in *team*.
Tuttavia, ho compreso di non aver sfruttato appieno questa opportunità, a causa di scelte errate effettuate durante l'esecuzione del progetto pratico associato al corso.
Tali scelte, dettate forse da inesperienza o da una sottovalutazione dell'importanza di alcuni aspetti del corso, mi hanno impedito di trarre tutto il valore che avrei potuto ottenere.

Lo *stage*, però, mi ha dato l'occasione di rimediare a queste mancanze, offrendomi uno spazio in cui applicare e migliorare concretamente le conoscenze acquisite all'interno del corso sopracitato.

Ritengo quindi che l'università mi abbia dato delle buonissime basi teoriche, ma è stato solo grazie all'esperienza di *stage* che ho potuto trasformare queste basi in competenze pratiche e operative, fondamentali per affrontare il mondo del lavoro.

4.4 Considerazioni finali

Concludendo, ritengo l'esperienza di *stage* presso *Zero12* estremamente positiva, sia per l'argomento trattato che per il risultato raggiunto.
In particolare, lavorare su un tema innovativo e di grande rilevanza come la *Generative AI*, combinato con l'utilizzo dei servizi AWS, mi ha permesso di acquisire competenze tecniche avanzate e di esplorare tecnologie all'avanguardia.

Sono riuscito a creare un progetto completo, che ha superato le mie stesse aspettative. Questo risultato non solo mi ha dato grande soddisfazione personale, ma ha anche consolidato la mia fiducia nelle mie capacità di affrontare sfide complesse e di portare a termine attività ambiziose.

Anche l'ambiente di lavoro presso *Zero12* è stato un elemento chiave per il successo di questa esperienza.
Ho trovato un contesto altamente stimolante, caratterizzato da un elevato livello di professionalità e da una cultura aziendale orientata all'innovazione e alla sperimentazione.
La disponibilità ed il supporto dei colleghi, uniti a una costante attenzione alla formazione e al miglioramento, hanno reso questo percorso formativo non solo proficuo, ma anche particolarmente gratificante.

In conclusione, considero questo *stage* un'esperienza fondamentale per la mia crescita personale e professionale, che ha arricchito il mio bagaglio di conoscenze e mi ha fornito strumenti preziosi per il mio futuro lavorativo.

Acronimi e abbreviazioni

AI Artificial Intelligence. 15, 17, 33, 82

API Application Program Interface. vi, 10, 15, 16, 18, 19, 21, 28, 32, 33, 36–38, 44, 55–60, 65, 73, 74, 76, 82

AWS Amazon Web Services. 1, 15–17, 23, 25, 29, 30, 33, 35, 37, 38, 40, 77, 79, 80, 82

CI/CD Continuous Integration/Continuous Deployment. 16, 82

CSS Cascading Style Sheets. 7, 11, 83

DI Dependency Injection. 56, 83

DTO Data Transfer Object. 56, 60

HTML HyperText Markup Language. 7, 9, 62, 83

HTTP Hypertext Transfer Protocol. 10, 12, 18, 55, 60, 83

IDE Integrated Development Environment. 4, 22, 83

ITS Issue Tracking System. 3, 84

JWT JSON Web Token. 16, 84

LLM Large Language Model. vi, 15, 27, 31, 32, 35–37, 49–51, 60, 61, 64, 74–76, 78, 84

NPM *Node Package Manager*. 18

ODM Object Data Modeling. 14, 54, 84

PR Pull Request. 4, 20, 84

RAG Retrieval Augmented Generation. 15, 84

SPA Single Page Application. 85

UI User Interface. 53, 85

UML Unified Modeling Language. 19, 85

Glossario

AI Il termine *Artificial Intelligence* si riferisce ad un campo dell'informatica che si occupa dello sviluppo di algoritmi e modelli che permettono ai computer di eseguire compiti che richiedono intelligenza umana. Questi algoritmi e modelli sono in grado di apprendere dai dati e di migliorare le proprie prestazioni nel tempo. 49, 81

API Il termine *Application Programming Interface API* indica ogni insieme di procedure disponibili al programmatore, di solito raggruppate a formare un set di strumenti specifici per l'espletamento di un determinato compito all'interno di un certo programma. La finalità è ottenere un'astrazione, di solito tra l'hardware e il programmatore o tra software a basso e quello ad alto livello semplificando così il lavoro di programmazione. 11, 81

API RESTful Il termine *API RESTful* si riferisce ad un'API che segue i principi dell'architettura REST. Questo tipo di API permette di accedere e manipolare le risorse tramite richieste HTTP, utilizzando i metodi standard di HTTP (GET, POST, PUT, DELETE). 11, 12, 33, 37, 38, 54

AWS *Amazon Web Services* è una piattaforma di servizi cloud offerta da Amazon. Questi servizi sono divisi in diverse categorie, tra cui calcolo, storage, database, analisi, machine learning, sicurezza, sviluppo di applicazioni, mobile, IoT, IA, AR e VR, media e sviluppo di giochi. AWS offre oltre 200 servizi completi da data center in tutto il mondo. 1, 81

back-end Il termine *back-end* si riferisce alla parte di un'applicazione che gestisce i dati e la logica di business dell'applicazione. Questa parte dell'applicazione è responsabile della gestione dei dati, della sicurezza e delle prestazioni dell'applicazione. v, 10–12, 18, 21, 33, 35–38, 40, 55, 56, 58–60, 63–66, 77

Branch Il termine *branch* si riferisce ad una copia separata del codice sorgente di un progetto. Questa copia separata viene utilizzata per sviluppare nuove funzionalità o correggere bug senza influenzare il codice principale (develop/main) del progetto. 4, 20

Chromium Il termine *Chromium* si riferisce ad un progetto *open source* che fornisce la base per molti *browser web* moderni, tra cui *Google Chrome, Microsoft Edge e Opera*. Questo progetto è sviluppato da *Google* ed offre un'implementazione veloce e sicura del motore di *rendering WebKit*. 14

CI/CD Il termine *Continuous Integration/Continuous Deployment* si riferisce ad un approccio di sviluppo *software* che prevede l'integrazione continua delle modifiche

al codice ed il rilascio continuo delle nuove versioni dell'applicazione. Questo approccio permette di ridurre i tempi di sviluppo e di rilascio dell'applicazione. 81

Container Il termine *container* si riferisce ad un'unità software che contiene un'applicazione e tutte le sue dipendenze. Questa unità software è isolata dal sistema operativo e può essere eseguita su qualsiasi sistema che supporta i container. 18, 59

CSS Il termine *Cascading Style Sheets* si riferisce ad un linguaggio di stile utilizzato per definire l'aspetto e la formattazione di una pagina *web*. Questi fogli di stile permettono di separare la struttura e il contenuto di una pagina *web* dalla sua presentazione. 81

DI Il termine *Dependency Injection* si riferisce ad un *design pattern* che permette di separare la creazione e la gestione delle dipendenze di un oggetto. Questo *design pattern* permette di ridurre la dipendenza tra i componenti del sistema e di rendere il codice più flessibile e riutilizzabile. 56, 81

DTO Il termine *Data Transfer Object* si riferisce ad un oggetto che viene utilizzato per trasferire dati tra le componenti di un sistema. Questo oggetto contiene solo i dati necessari per la comunicazione tra i componenti e non contiene alcuna logica di *business*. 59, 81

front-end Il termine *front-end* si riferisce alla parte di un'applicazione che interagisce con l'utente. Questa parte dell'applicazione è responsabile della presentazione dei dati all'utente e della gestione delle interazioni con l'utente. v, 9–12, 18, 21, 33, 35, 40, 54, 55, 57, 58, 60, 61, 63, 65, 77, 78

Generative AI Il termine *Generative AI* si riferisce ad un campo dell'intelligenza artificiale che si occupa dello sviluppo di algoritmi e modelli che permettono ai computer di generare contenuti in modo automatico. Questi algoritmi e modelli sono in grado di generare testo, immagini, suoni e video in modo autonomo. 17, 25, 31, 38, 40, 75, 77, 79, 80

Git Flow Il termine *Git Flow* si riferisce a un modello di *branching* per il controllo delle versioni con *Git*. Questo modello prevede l'utilizzo di due branch principali, *master* e *develop*, e di branch di *feature*, *release* e *hotfix* per organizzare il lavoro di sviluppo. 34, 35, 57

HTML Il termine *HyperText Markup Language* si riferisce ad un linguaggio di *markup* utilizzato per creare pagine *web*. Questo linguaggio permette di definire la struttura e il contenuto di una pagina *web* utilizzando *tag* ed attributi. 7, 81

HTTP Il termine *Hypertext Transfer Protocol* si riferisce ad un protocollo di comunicazione utilizzato per trasferire informazioni tra il client e il server su Internet. Questo protocollo è basato sul concetto di richiesta e risposta, dove il client invia una richiesta al server e il server risponde con i dati richiesti. 81

IDE Un IDE (Integrated Development Environment) è un software che fornisce un ambiente di sviluppo integrato per la scrittura, il debugging ed il testing di codice. Questo software fornisce agli sviluppatori tutti gli strumenti necessari per scrivere e testare il codice in un unico ambiente. 4, 81

ITS Un *Issue Tracking System* è un software che permette di gestire e monitorare le attività, i problemi e le richieste di assistenza dei clienti. Questo sistema permette di tenere traccia di tutte le attività, assegnarle ai membri del team e monitorare il loro stato. 81

JWT Il termine *JSON Web Token* si riferisce ad uno standard aperto che definisce un modo compatto e autonomo per rappresentare informazioni tra due parti. Questo standard è utilizzato per trasmettere informazioni tra il client e il server in modo sicuro e affidabile. 81

LLM Il termine *Large Language Model* si riferisce ad un modello di linguaggio che utilizza tecniche di apprendimento automatico per generare testo in modo automatico. Questi modelli sono addestrati su grandi quantità di testo per apprendere la struttura e il significato del linguaggio naturale. 26, 27, 81

Machine Learning Il termine *Machine Learning* si riferisce ad un campo dell'intelligenza artificiale che si occupa dello sviluppo di algoritmi e modelli che permettono ai computer di apprendere dai dati e di migliorare le proprie prestazioni nel tempo. 17

ODM Il termine *Object Data Modeling* si riferisce al processo di progettazione e creazione di modelli di dati basati su oggetti. Questo approccio permette di rappresentare i dati in modo più naturale e flessibile rispetto ai tradizionali modelli di dati relazionali. 59, 81

PR Il termine *Pull Request* si riferisce ad una richiesta di un membro del team di unire un proprio branch (sia esso di feature o bugix) con il codice principale (develop/main) del progetto. Questa richiesta viene utilizzata per revisionare il codice, discutere le modifiche e garantire che il codice sia conforme agli standard del progetto prima di unirlo al codice principale. 81

Product Backlog Il *Product Backlog* è una lista di tutte le funzionalità, i requisiti, le modifiche e le correzioni che devono essere fatte ad un prodotto. Questa lista è dinamica e viene aggiornata costantemente, in modo da riflettere le esigenze del cliente. 2, 45

Prompt Il *Prompt* è una frase o un paragrafo che viene utilizzato per guidare un sistema di Intelligenza Artificiale nella creazione di contenuti. vi, 25, 47, 51, 67, 71, 72, 84

Prompt Engineering Il termine *Prompt Engineering* si riferisce al processo di progettazione e creazione di prompt per un modello di linguaggio. Questo processo è fondamentale per ottenere risultati accurati e coerenti da un modello di linguaggio. 15

RAG Il termine *Retrieval Augmented Generation* si riferisce ad un approccio ibrido che combina tecniche di generazione di testo e di recupero di informazioni. Questo approccio permette di generare testo in modo automatico utilizzando informazioni recuperate da una base di conoscenza. 81

SPA Il termine *Single Page Application* si riferisce ad un'applicazione web che carica una sola pagina HTML e aggiorna dinamicamente il contenuto della pagina senza ricaricare l'intera pagina. Questo approccio permette di creare applicazioni web più veloci e reattive. 9, 81

sprint Il termine *sprint* si riferisce a un periodo di tempo delimitato, di solito una o due settimane, durante il quale il *team* di sviluppo lavora per completare un insieme di obiettivi prefissati. 34, 41, 42, 44, 45, 65, 79

token Il termine *token* si riferisce ad una sequenza di caratteri che rappresenta un'unità di informazione. Questi possono essere parole intere, parti di parole, o anche singoli caratteri, a seconda del modello e della sua configurazione. 27, 28

UI Il termine *User Interface* si riferisce alla parte di un'applicazione che interagisce con l'utente. Questa parte dell'applicazione è responsabile della presentazione dei dati all'utente e della gestione delle interazioni con l'utente. 81

UML Il termine *UML, Unified Modeling Language* è un linguaggio di modellazione e specifica basato sul paradigma object-oriented. L'*UML* svolge un'importantissima funzione di "lingua franca" nella comunità della progettazione e programmazione a oggetti. Gran parte della letteratura di settore usa tale linguaggio per descrivere soluzioni analitiche e progettuali in modo sintetico e comprensibile a un vasto pubblico. 81

User Stories Le *User stories* rappresentano una pratica *Agile*, che viene utilizzata per comprendere le esigenze dell'utente, mediante una descrizione informale, semplice e concisa della funzionalità.. 2, 32, 34, 41, 42, 45–47, 76

Bibliografia

Siti web consultati

Artificial Intelligence. URL: https://https://www.ibm.com/it-it/topics/artificial-intelligence/.

AWS Amplify. URL: https://aws.amazon.com/amplify/.

AWS Bedrock. URL: https://aws.amazon.com/it/bedrock/ (cit. a p. 27).

AWS Cognito. URL: https://aws.amazon.com/cognito/.

AWS S3. URL: https://aws.amazon.com/s3/.

Confluence. URL: https://www.atlassian.com/software/confluence.

Docker. URL: https://www.docker.com/.

Git Flow. URL: https://www.atlassian.com/git/tutorials/comparing-workflows/gitflow-workflow (cit. a p. 34).

GitHub. URL: https://github.com/.

Introduction to LLM. URL: https://www.hpe.com/it/it/what-is/large-language-model.html (cit. a p. 26).

JavaScript vs TypeScript. URL: https://medium.com/geekculture/typescript-vs-javascript-e5af7ab5a331 (cit. a p. 8).

Jira. URL: https://www.atlassian.com/software/jira.

JWT Token. URL: https://jwt.io/.

Key differences between main LLMs. URL: https://ai-pro.org/learn-ai/articles/a-comprehensive-comparison-of-all-llms/ (cit. a p. 27).

Key features of LangChain. URL: https://datasciencedojo.com/blog/what-is-langchain/ (cit. a p. 16).

LangChain. URL: https://https://js.langchain.com/docs/introduction/.

Language Model. URL: https://https://www.hpe.com/emea_europe/en/what-is/large-language-model.html/.

LLM parameters. URL: https://ivibudh.medium.com/a-guide-to-controlling-llm-model-output-exploring-top-k-top-p-and-temperature-parameters-ed6a31313910 (cit. a p. 26).

Manifesto Agile. URL: http://agilemanifesto.org/iso/it/.

Medium. URL: https://medium.com/.

MongoDB. URL: https://www.mongodb.com/.

MongoDB Compass. URL: https://www.mongodb.com/products/compass.

Mongoose. URL: https://mongoosejs.com/.

MoSCoW Method. URL: https://en.wikipedia.org/wiki/MoSCoW_method/ (cit. a p. 46).

NestJS. URL: https://nestjs.com/.

NPM. URL: https://www.npmjs.com/.

Postman. URL: https://www.postman.com/.

React. URL: https://reactjs.org/.

React PDF. URL: https://www.npmjs.com/package/react-pdf/.

Retrieval Augmented Generation. URL: https://aws.amazon.com/it/what-is/retrieval-augmented-generation/ (cit. a p. 15).

Scrum. URL: https://www.scrum.org/.

Slack. URL: https://slack.com/.

SQL vs NoSQL. URL: https://gowithcode.com/sql-vs-nosql (cit. a p. 13).

Swagger. URL: https://swagger.io/.

Update of Sonnet 3.5. URL: https://www.anthropic.com/news/3-5-models-and-computer-use (cit. a p. 37).

User Stories Mapping. URL: https://jpattonassociates.com/story-mapping/ (cit. a p. 45).

V-Model. URL: https://www.geeksforgeeks.org/software-engineering-sdlc-v-model/ (cit. a p. 65).

Visual Studio Code. URL: https://code.visualstudio.com/.

Zero12. URL: https://www.zero12.it/.