**Università degli Studi di Padova**
DIPARTIMENTO DI MATEMATICA "TULLIO LEVI-CIVITA"
CORSO DI LAUREA IN INFORMATICA


# Sviluppo di un sistema di Skills Intelligence: pipeline di estrazione NLP e strumenti di validazione delle competenze

*Tesi di Laurea*

*Relatore*
Prof. Francesco Ranzato

*Laureando*
Leonardo Salviato
Matricola 2101086

Anno Accademico 2025-2026

*"Se fai solo quello che sai fare, non sarai mai più di quello che sei ora."*

— Kung Fu Panda 3, Maestro Shifu

# Ringraziamenti

Al termine di questo percorso, ci tengo a ringraziare il mio relatore, il Prof. Francesco Ranzato, per la guida e la costante disponibilità offerte durante la stesura di questo elaborato. Desidero inoltre esprimere la mia gratitudine all'azienda EggOn e a tutto il suo team: grazie per l'ospitalità e per il sincero interesse dimostrato nei confronti del mio lavoro.

In questi tre anni di università ho incontrato persone straordinarie che hanno reso questa esperienza stimolante e divertente, accompagnandomi in ogni fase, tra difficoltà e soddisfazioni. Un ringraziamento speciale va a voi, che siete stati con me durante ogni lezione, ogni esame, ogni sessione e ogni viaggio: senza di voi non sarebbe stata la stessa cosa.

Anche se non ci vediamo spesso, non importa: tutto ciò che abbiamo costruito e vissuto assieme è prova dell'amicizia che ci lega. Non c'è bisogno di uscire o di sentirci ogni giorno; ogni esperienza, risata e avventura condivisa è sempre con me e costituisce una parte fondamentale della persona che sono oggi. Grazie a voi, che avete reso speciali i periodi più felici della mia vita.

Ci siamo conosciuti quando eravamo bambini, pensavamo solo a uscire e a divertirci e, adesso che siamo cresciuti, in fondo non è cambiato molto: abbiamo solo qualche dovere in più e, di tanto in tanto, ci tocca comportarci da adulti. Non riesco a immaginare una vita in cui non siate presenti e, anche se il futuro ci porterà in posti diversi o lontani, so che quando conterà davvero ci saremo sempre l'uno per l'altro. Un ringraziamento profondo va a voi, che siete e sarete sempre la mia costante.

Infine, non basterebbero le parole per esprimere la gratitudine che provo nei vostri confronti: vi siete presi cura di me e mi avete dato tutto ciò che potevate, senza mai chiedere nulla in cambio. Alla mia famiglia, ai miei genitori, che mi hanno cresciuto e amato incondizionatamente, e a mio fratello, che da sempre è il mio punto di riferimento, dico semplicemente grazie. Vi amo dal profondo del cuore: grazie per essere stati la mia motivazione più grande.

Padova, Luglio 2026 *Leonardo Salviato*

# Abstract

Questo lavoro di tesi presenta lo sviluppo di un prototipo di Skills Intelligence[G] per la piattaforma di Human Resources (HR) Nexum. Il componente più importante del sistema è una pipeline di Natural Language Processing (NLP[G]) progettata per automatizzare l'estrazione di informazioni da documenti non strutturati, come i CV[G], riconoscendo e normalizzando le skill rispetto a un dizionario standard. L'architettura integra inoltre meccanismi di validazione lato HR per garantire la creazione di profili utente dinamici e sempre aggiornati. I dati elaborati alimentano infine uno "Skill Radar[G]", un'interfaccia grafica che permette di analizzare le competenze aggregate, identificare i gap formativi e ottimizzare l'allocazione dei talenti, supportando il management in decisioni strategiche data-driven.

# Indice

# Elenco delle figure

# Elenco delle tabelle

# Elenco degli acronimi

**ACID:** Atomicity, Consistency, Isolation, Durability

**AI:** Artificial Intelligence

**API:** Application Programming Interface

**AR/VR:** Augmented Reality / Virtual Reality

**AWS:** Amazon Web Services

**BRD:** Business Requirements Document

**CA:** Criteri di Accettazione

**CEO:** Chief Executive Officer

**CRUD:** Create, Read, Update, Delete

**CV:** Curriculum Vitae

**DI:** Dependency Injection

**DOM:** Document Object Model

**E2E:** End-to-End

**ER:** Entità-Relazione

**ESCO:** European Skills, Competences, Qualifications and Occupations

**GDPR:** General Data Protection Regulation

**GUI:** Graphical User Interface

**HR:** Human Resources

**HTML:** HyperText Markup Language

**HTTP:** Hypertext Transfer Protocol

**IoT:** Internet of Things

**JSON:** JavaScript Object Notation

**LLM:** Large Language Model

**MVC:** Model-View-Controller

**MVVM:** Model-View-ViewModel

**NER:** Named Entity Recognition

**NLP:** Natural Language Processing

**ONA:** Organizational Network Analysis

**ORDBMS:** Object-Relational Database Management System

**ORM:** Object-Relational Mapping

**REST:** Representational State Transfer

**RoR:** Ruby on Rails

**RxJS:** Reactive Extensions for JavaScript

**SPA:** Single Page Application

**SQL:** Structured Query Language

**UAT:** User Acceptance Testing

**UI:** User Interface

**UML:** Unified Modeling Language

**URI:** Uniform Resource Identifier

**URL:** Uniform Resource Locator

# Capitolo 1

# Introduzione

## 1.1 Il contesto aziendale

### 1.1.1 Presentazione dell'azienda ospitante

Eggon è un'azienda specializzata nella progettazione e nello sviluppo di soluzioni software. Il nucleo operativo si concentra sull'implementazione di ecosistemi digitali in molteplici settori, tra cui: Applicazioni Mobile, Piattaforme Web, Digital Health, AR/VR[G], IoT[G], Gaming e Fintech. L'obiettivo aziendale è fornire strumenti tecnologici avanzati a imprese di diverse dimensioni, supportandone i processi di trasformazione e diversificazione digitale.

### 1.1.2 Presentazione della piattaforma Nexum

Nexum è una piattaforma gestionale e HR con architettura per contesti *Enterprise*, il cui scopo primario è fornire alle aziende uno strumento centralizzato, sicuro ed efficace per orchestrare le informazioni, monitorare le performance e mappare le attività quotidiane degli utenti. Sviluppata seguendo un'architettura *multi-tenant*[G], la piattaforma garantisce un rigoroso isolamento dei dati, consentendo a diverse organizzazioni di operare simultaneamente sul medesimo software in totale sicurezza.

Nexum è concepita come un prodotto dinamico e in continua evoluzione, supportato da un team di sviluppo che lavora per affinarne le funzionalità, le interfacce e la logica di business. Grazie al suo design modulare, la piattaforma è intrinsecamente flessibile e scalabile: riesce infatti ad adattarsi sia alle esigenze operative delle piccole-medie imprese, sia ai complessi flussi organizzativi tipici delle grandi multinazionali, coprendo l'intero ciclo di vita del dipendente all'interno dell'azienda.

## 1.2 Descrizione del problema

Nonostante Nexum sia ricca di funzionalità per la gestione amministrativa e il tracciamento delle attività del personale, la piattaforma presenta una limitazione significativa:

l'assenza di un sistema capace di mappare, quantificare e mostrare le competenze dei dipendenti in modo dinamico e standardizzato.

Nell'attuale panorama aziendale, la gestione delle competenze (*Skills Management*) non è più una mera attività di archiviazione, ma un elemento fondamentale per la competitività. Storicamente, la conoscenza del capitale umano aziendale si basa su documenti non strutturati (come i Curriculum Vitae), i quali diventano statici e obsoleti nel momento stesso in cui vengono caricati a sistema. Questa frammentazione e la mancanza di un dizionario tassonomico unificato comportano diverse criticità operative:

- **Difficoltà nel Workforce Planning[G]**: Quando è necessario avviare un nuovo progetto, i Project Manager riscontrano difficoltà nell'individuare rapidamente le risorse interne più idonee, dovendo basarsi sullo storico aziendale o su ricerche manuali inefficaci.
- **Carenza di Analisi dei Gap (Skill Gap[G] Analysis)**: Senza una mappatura numerica e standardizzata tra le competenze possedute dai dipendenti e i requisiti attesi dai profili di ruolo aziendali (*Role Profiles*), le divisioni HR non riescono a identificare tempestivamente le aree di debolezza.
- **Formazione non ottimizzata**: L'assenza di metriche precise impedisce la creazione di percorsi formativi mirati o l'instaurazione di programmi di *peer-to-peer learning* (affiancamento), limitando la capacità dell'azienda di valorizzare e trattenere il proprio talento.

In sintesi, la mancanza di un motore di *Skill Intelligence* costringe il management a prendere decisioni basate su intuizioni e dati frammentati, rallentando la mobilità interna e disperdendo il potenziale inespresso della forza lavoro.

## 1.3 Obiettivi e contributi della tesi

Per superare i limiti evidenziati, questo documento propone la progettazione e lo sviluppo di un modulo avanzato di Skill Intelligence, adeguatamente integrato nel progetto Nexum. L'obiettivo centrale è la transizione da un modello di archiviazione statica a una gestione dinamica, proattiva e data-driven delle competenze.

In questo contesto, il contributo progettuale e ingegneristico si è articolato su quattro pilastri funzionali:

1. **Standardizzazione tassonomica**: Implementazione di un dizionario unificato delle competenze, integrando skill personalizzate aziendali con framework ontologici standardizzati a livello europeo come l'infrastruttura dell' "European Skills, Competences, Qualifications and Occupations" (ESCO[G]), garantendo così interoperabilità, normalizzazione e supporto multilingua nativo.

2. **Pipeline NLP e intelligenza artificiale**: Sviluppo di un motore di estrazione automatizzata (*CvAnalyzer*) basato su Large Language Models (LLM[G]) forniti dall'infrastruttura Cloud. Il sistema elabora i documenti non strutturati estraendo le competenze tramite *Named Entity Recognition* e allineandole semanticamente al dizionario aziendale grazie ad algoritmi di *Suggestion Matching* e telemetria ad auto-apprendimento.

3. **Motore algoritmico e matching**: Realizzazione di una suite di servizi dedicata all'incrocio relazionale dei dati. Il motore calcola in tempo reale gli *Skill Gap*[G], supporta l'assegnazione automatica di percorsi di tutoraggio tra colleghi e suggerisce l'allocazione ottimale dei talenti sui nuovi progetti aziendali (*Internal Mobilization*).

4. **Visualizzazione ecosistemica**: Sviluppo di interfacce complesse e dashboard interattive. I dati elaborati alimentano lo *Skill Radar* e svariati grafi relazionali (come *Sunburst Chart*[G] e grafi di ecosistema dei progetti), riducendo la complessità dei Big Data HR e fornendo al management rappresentazioni visive immediate e azionabili.

Il risultato finale è un prototipo che non solo automatizza il reperimento delle informazioni, ma trasforma l'organigramma aziendale in una rete di competenze navigabile e interrogabile.

## 1.4 Struttura dell'elaborato

Di seguito viene presentata l'articolazione dell'elaborato:

- Capitolo 1: Introduzione.
- Capitolo 2: Background Tecnologico e Teorico - Panoramica sulle tecnologie NLP, NER[G], e stack tecnologico utilizzato.
- Capitolo 3: Struttura e organizzazione del progetto - Descrizione della metodologia di lavoro e pianificazione del lavoro.
- Capitolo 4: Analisi dei requisiti - Descrizione degli attori, dei casi d'uso dell'applicativo e dei requisiti necessari.
- Capitolo 5: Progettazione del sistema - Descrizione della progettazione dell'applicativo nei suoi vari componenti.
- Capitolo 6: Verifica e validazione - Metodologia di testing, valutazione delle performance del *parsing* e verifica della coerenza grafica e dei dati.
- Capitolo 7: Conclusioni - Riflessioni sui risultati raggiunti, maturazione professionale e sviluppi futuri.
- Bibliografia e Glossario - Riferimenti bibliografici e definizione dei termini chiave utilizzati nel documento.

# Capitolo 2

# Background tecnologico e teorico

## 2.1 Elaborazione del linguaggio naturale

L'elaborazione del linguaggio naturale (NLP) è un sottocampo dell'informatica e dell'intelligenza artificiale (AI[G]) che utilizza il machine learning, la linguistica computazionale, la modellazione statistica e altre tecniche per permettere ai computer di comprendere, interpretare e generare il linguaggio umano. Negli ultimi anni, l'NLP ha visto un rapido sviluppo grazie all'avvento dei Large Language Models, che hanno permesso a diversi sistemi di elaborare e produrre testo in modo più naturale e contestuale. Questi modelli sono stati addestrati su enormi quantità di dati testuali e sono in grado di eseguire una vasta gamma di compiti NLP, tra cui la traduzione automatica, la risposta a domande, la generazione di testo e l'estrazione di informazioni. Nel contesto della tesi, l'NLP è fondamentale per sviluppare una pipeline efficace per l'estrazione delle competenze dai CV dei dipendenti e per l'analisi delle informazioni estratte in contesti progettuali [20].

### 2.1.1 Named Entity Recognition ed estrazione di informazioni

La Named Entity Recognition (NER), nota anche come *entity extraction* o *entity chunking*, è uno dei principali rami dell'Elaborazione del Linguaggio Naturale (NLP) nel campo dell'Information Extraction. Il suo scopo fondamentale è processare testi non strutturati per individuare, isolare e classificare elementi informativi chiave (le "entità nominate") all'interno di categorie semantiche predefinite, come nomi di persone, organizzazioni, luoghi, date o concetti tecnici specifici [17].

Nel contesto dell'ingegneria delle Risorse Umane e dello *Skills Management*, la NER assume un ruolo centrale attraverso il processo di *Resume Parsing* (analisi automatizzata dei CV). I curriculum vitae sono per natura documenti non strutturati, redatti in formati discorsivi, impaginazioni variabili e stili linguistici eterogenei. Attraverso la NER, il sistema è in grado di leggere questo testo libero e trasformarlo in un set di dati strutturato (tipicamente in formato JSON[G]), isolando l'anagrafica del candidato, il percorso accademico, le esperienze lavorative e, in particolare, le competenze.

### 2.1.2 Modello utilizzato: Claude Haiku 4.5

Per il progetto è stato utilizzato il modello Claude Haiku 4.5 di Anthropic, un LLM avanzato che offre capacità di comprensione e generazione del linguaggio naturale. Questo modello è stato scelto in virtù della sua velocità di inferenza superiore alla media, per il suo costo computazionale contenuto e la sua ottimizzazione nei compiti che richiedono bassa latenza e l'elaborazione di grandi volumi di dati. La famiglia di modelli Anthropic è stata scelta per la sua capacità di analizzare il file fornito considerando simultaneamente i due aspetti principali: il testo scritto del curriculum e la sua struttura. Questa caratteristica si rivela necessaria nell'elaborazione necessario di documenti non strutturati dove le informazioni chiave possono essere scritte utilizzando formati diversi [8].

## 2.2 Stack tecnologico e strumenti

Le scelte tecnologiche alla base dello sviluppo del modulo *Skill Intelligence* sono state fortemente guidate dalla necessità di innestare le nuove funzionalità all'interno dell'ecosistema *legacy* della piattaforma Nexum. Di conseguenza, gran parte dello stack riflette i vincoli architetturali imposti dall'azienda, i quali hanno tuttavia garantito un ambiente di sviluppo maturo e accelerato l'implementazione del progetto.

### 2.2.1 Framework e linguaggi di sviluppo

#### Ruby on Rails

Ruby on Rails è un framework full-stack basato sul linguaggio di programmazione orientato agli oggetti Ruby, configurato in questo progetto esclusivamente in modalità API[G]-only [31, 30].
**Motivazione e ruolo:** L'impiego di Rails ha rappresentato un vincolo aziendale imprescindibile per mantenere la compatibilità con il *backend* esistente. Nel contesto del tirocinio, il framework ha permesso uno sviluppo efficiente grazie all'ORM[G] Active Record[G], consentendo di modellare agevolmente il database e di esporre rapidamente le API RESTful necessarie. Questo approccio ha garantito una netta separazione delle responsabilità, mantenendo il server focalizzato unicamente sull'orchestrazione delle logiche di business HR.

#### Angular e TypeScript

Angular è un framework strutturato per la creazione di Single Page Application (SPA[G]), basato nativamente su TypeScript, un superset di JavaScript a tipizzazione statica [7, 33].
**Motivazione e ruolo:** L'adozione di Angular è stata dettata dalla necessità di mantenere la conformità con lo stack frontend aziendale. TypeScript si è rivelato fondamentale per governare la complessità dei dati scambiati con il server, riducendo gli errori a runtime tramite la tipizzazione rigorosa. Angular è stato impiegato per sviluppare l'intera interfaccia

utente del nuovo modulo: dalla gestione dei form reattivi per la validazione delle competenze, fino all'integrazione delle librerie di visualizzazione dati necessarie per il rendering dello *Skill Radar* e dei grafi interattivi.

### 2.2.2 Infrastruttura AWS

#### Amazon Bedrock

Amazon Bedrock è un servizio AWS completamente gestito che fornisce accesso a modelli fondativi (FM) di Intelligenza Artificiale generativa [6].
**Motivazione e ruolo:** Questa tecnologia è stata selezionata appositamente per delegare le complesse operazioni di estrazione NLP dai documenti. In alternativa allo sviluppo di modelli computazionalmente onerosi in Python, come inizialmente ipotizzato, la scelta è ricaduta sull'invocazione tramite API del modello AI. Questa decisione ingegneristica ha permesso di trasformare il problema dell'estrazione delle *skill* in un'attività di *Prompt Engineering*, garantendo un'elevata precisione semantica, costi infrastrutturali contenuti e tempi di esecuzione del *parsing* dei CV inferiori ai dieci secondi.

#### Amazon S3

Amazon Simple Storage Service (Amazon S3[G]) è un servizio cloud standard per l'archiviazione a oggetti [4].
**Motivazione e ruolo:** La scelta di S3 garantisce la continuità con l'infrastruttura Cloud già in uso in azienda. Nel dominio di questo progetto, S3 svolge il ruolo cruciale di archivio temporaneo e sicuro per i Curriculum Vitae (PDF, DOCX) caricati dai dipendenti. Ha fornito un punto di appoggio scalabile e conforme ai requisiti di sicurezza per conservare i file prima e durante l'elaborazione da parte del *worker* asincrono incaricato di processarli tramite l'IA.

### 2.2.3 Database e tassonomia

#### PostgreSQL

PostgreSQL è un potente sistema di gestione di database relazionale a oggetti (ORDBMS), rinomato per l'integrità transazionale [26].
**Motivazione e ruolo:** Essendo uno standard imposto dall'architettura base di Nexum, l'uso di PostgreSQL si è confermato indispensabile per gestire l'isolamento dei dati multi-tenant (ACID[G]). Nel lavoro di tesi, PostgreSQL è stato sfruttato intensivamente per una sua specifica peculiarità: il supporto nativo e indicizzato al formato `JSONB`. Tale caratteristica è stata architetturalmente fondamentale per il modulo *Skill Intelligence*, in quanto ha permesso il salvataggio di dati complessi a seguito delle validazioni HR.

### ESCO (European Skills, Competences, Qualifications and Occupations)

ESCO è il dizionario standardizzato e multilingue della Commissione Europea per la classificazione delle professioni e delle competenze [14].

**Motivazione e ruolo:** A differenza delle tecnologie precedenti, ESCO rappresenta una scelta metodologica di dominio, introdotta per risolvere il problema dell'ambiguità semantica. È stato adottato come dizionario iniziale per il processo di normalizzazione dei dati.

# Capitolo 3

# Struttura e organizzazione del progetto

## 3.1 Norme di progetto e metodologia di lavoro

Il progetto ha avuto inizio con la consegna di un Business Requirement Document (BRD[G]) da parte dell'azienda, che ha delineato in modo chiaro e dettagliato le funzionalità richieste già definite nel piano di lavoro.

### 3.1.1 Modello di sviluppo

La metodologia scelta per lo svolgimento di questo progetto è stato il modello Agile. Il modello di sviluppo Agile segue un approccio iterativo e incrementale, utilizzato per la gestione dei progetti e per lo sviluppo software. A differenza di altri paradigmi tradizionali come il modello a cascata, caratterizzati da una struttura rigorosamente sequenziale, l'approccio Agile scompone il progetto in cicli di sviluppo brevi e flessibili, chiamati "iterazioni" o "sprint". Queste caratteristiche rendono questo modello adatto a progetti complessi e promuovono una stretta collaborazione tra i membri del team e il cliente [1].

### 3.1.2 Riunioni

Per garantire un allineamento costante con il tutor aziendale, è stato istituito uno *standup meeting* giornaliero, durante il quale venivano discussi i progressi, le sfide incontrate e chiariti eventuali dubbi per i passi successivi. A seguito della consegna dei primi deliverable, si è tenuto un *meeting* di revisione con il tutor aziendale e il CEO[G] dell'azienda, promotore dell'iniziativa, per valutare i risultati raggiunti e pianificare le fasi successive del progetto. Successivamente alla prima fase, si è stabilito di istituire un *meeting* settimanale con i referenti oltre ai meeting giornalieri, per discutere in modo più approfondito i progressi.

### 3.1.3 Comunicazioni

La comunicazione con il tutor aziendale e il team di sviluppo è stata gestita principalmente attraverso strumenti di messaggistica istantanea (Telegram), che hanno permesso di mantenere un flusso di comunicazione rapido e diretto, facilitando la risoluzione tempestiva di eventuali problemi o dubbi. Per la gestione di eventi e scadenze importanti è stato invece utilizzato Google Calendar, supportato da notifiche via email, che ha consentito di organizzare in modo efficace il tempo e le attività del progetto.

## 3.2 Pianificazione settimanale

La pianificazione dello stage, strutturata su una durata di otto settimane, è stata ideata per coprire progressivamente le cinque fasi architetturali definite nel Business Requirements Document per l'evoluzione del sistema di Skill Intelligence all'interno della piattaforma Nexum. L'obiettivo della pianificazione è garantire un rilascio incrementale, dall'infrastruttura dati fino agli algoritmi di matching aziendale.

Una tabella esplicativa della pianificazione è consultabile nell'Appendice A.

### 3.2.1 Prima settimana: Analisi architetturale e fase 1 (Modello Dati)

La prima settimana è stata dedicata all'onboarding tecnologico e alla progettazione delle fondamenta del sistema:

- **Setup tecnologico**: configurazione dell'ambiente di sviluppo full-stack secondo i vincoli architetturali richiesti, basato su Ruby on Rails (RoR) per il backend, Angular per il frontend e infrastruttura AWS[G].
- **Progettazione Anagrafica Competenze (Fase 1)**: definizione e modellazione del database PostgreSQL per ospitare il nuovo modello dati delle skill, la tassonomia delle competenze e i relativi livelli di valutazione.
- **Gestione Backend HR**: avvio dello sviluppo delle API per la creazione delle skill e la loro associazione manuale agli utenti.

### 3.2.2 Seconda settimana: Sviluppo tassonomia e inizio Fase 2 (Parsing NLP)

Durante la seconda settimana, l'attività si è divisa tra il consolidamento del data model e l'inizio dello studio per l'automazione dell'estrazione:

- **Completamento Fase 1**: implementazione delle logiche di categorizzazione (hard skill[G], soft skill[G], linguaggi, ruoli) e delle relazioni gerarchiche tra le competenze nel dizionario standard.
- **Upload e gestione documenti (Fase 2)**: sviluppo del modulo per l'acquisizione e il caricamento di curriculum vitae nei formati supportati (PDF, DOC, DOCX).

- **Progettazione della pipeline NLP**: studio e definizione degli algoritmi di base per l'estrazione del testo grezzo dai documenti.

### 3.2.3 Terza settimana: Implementazione Data Extraction (Fase 2)

La terza settimana si è concentrata sulla realizzazione del motore di estrazione automatica delle skill:

- **Parsing e identificazione**: sviluppo dell'algoritmo per il riconoscimento automatico delle skill e l'estrazione degli anni e del contesto di esperienza lavorativa dal testo.
- **Normalizzazione dei dati**: implementazione delle regole di mapping per allineare i termini estratti dal CV al dizionario standard aziendale e logiche di deduplicazione.
- **Ottimizzazione delle performance**: raffinamento del codice di *parsing* per soddisfare il requisito di prestazione che impone un'elaborazione del CV in un tempo inferiore ai dieci secondi.

### 3.2.4 Quarta settimana: Profilo dinamico risorsa (Fase 3)

Nella quarta settimana, i dati estratti sono stati integrati nelle interfacce utente, creando un profilo interattivo e aggiornabile:

- **Interfaccia Skill-Based**: sviluppo su Angular delle viste per l'elenco delle competenze e la timeline[G] storica delle esperienze della singola risorsa.
- **Workflow di validazione HR**: creazione degli strumenti per la revisione e la conferma ufficiale dei dati da parte del reparto Risorse Umane, mitigando il rischio di dati incompleti o non accurati.

### 3.2.5 Quinta settimana: Skill Matching Engine (Fase 4)

La quinta settimana ha spostato il focus sull'analisi dei dati raccolti, introducendo logiche di elaborazione e restituzione visiva:

- **Generazione Skill Radar**: realizzazione della rappresentazione grafica multidimensionale delle competenze possedute dall'utente.
- **Analisi dei Gap Individuali**: sviluppo degli algoritmi per il confronto automatico tra le competenze attuali della risorsa e quelle teoricamente richieste dal suo ruolo.
- **Motore di Suggerimento**: implementazione iniziale di logiche predittive in grado di consigliare in automatico percorsi di formazione, opportunità di mobilità interna o attività di mentoring.

### 3.2.6 Sesta settimana: Mappatura progetti e matching (Fase 5)

Durante la sesta settimana è stato introdotto il modulo più avanzato del sistema, volto a supportare le decisioni strategiche di allocazione:

- **Definizione skill di progetto**: sviluppo di interfacce per permettere ai responsabili di inserire le competenze specifiche richieste per l'avvio di un nuovo progetto o per la copertura di un ruolo.
- **Ranking e Matching**: creazione del motore di ricerca interno per generare classifiche dei candidati più idonei e suggerire associazioni automatiche risorsa-progetto.

### 3.2.7 Settima settimana: Analisi organizzativa e integrazione

La penultima settimana è stata impiegata per completare le funzionalità di analisi macroscopica e unificare i diversi moduli sviluppati:

- **Gap Organizzativi (Fase 5)**: implementazione delle dashboard per la Direzione e i Talent Manager volte a identificare le carenze di competenze (skill shortage) sull'intera popolazione aziendale.
- **Sincronizzazione Real-Time**: ottimizzazione dell'architettura per garantire che l'aggiornamento dei dati sia propagato *near real-time* su tutte le viste e supporti configurazioni multi-organizzazione.

### 3.2.8 Ottava settimana: Validazione, collaudo e documentazione

L'ottava settimana è stata interamente dedicata alla validazione formale del software e alla stesura della documentazione finale:

- **Verifica Criteri di Accettazione (CA[G])**: esecuzione di test end-to-end per certificare il corretto funzionamento di tutte le componenti.
- **Risoluzione Anomalie**: attività di revisione manuale per mitigare eventuali imprecisioni residue introdotte dal modello di Machine Learning.
- **Documentazione**: stesura dell'elaborato di tesi, formalizzazione del Data Model finale e redazione dei manuali architetturali del sistema.

# Capitolo 4

# Analisi dei requisiti

## 4.1 Introduzione all'analisi dei requisiti

Il presente capitolo definisce e descrive con precisione i requisiti necessari per documentare in modo esaustivo la portata del progetto. Essendo un progetto di notevoli dimensioni, con numerosi requisiti e casi d'uso, è stato scelto di riportare solo i più significativi ed essenziali per mantenere intatta la leggibilità del documento.

## 4.2 Attori

### 4.2.1 Definizione

In fase di analisi dei requisiti, un attore rappresenta un ruolo interpretato da un utente umano, un dispositivo hardware o un altro sistema software che interagisce direttamente con il sistema in esame. L'identificazione degli attori è fondamentale per delineare i confini del sistema e per definire chi (o cosa) eseguirà i vari casi d'uso, determinando di conseguenza i permessi, le viste e i flussi operativi necessari.

### 4.2.2 Attori primari

Gli attori primari del progetto comprendono tutte le tipologie di utenti che dispongono di credenziali di accesso alla piattaforma Nexum e che interagiscono in modo diretto per inserire, gestire o consultare le informazioni relative alle competenze.

- **Responsabile HR / Talent Manager**: È l'utente principale lato *backoffice*. Si occupa della gestione del dizionario delle competenze, dell'upload dei CV e, soprattutto, della validazione manuale (human-in-the-loop[G]) delle competenze estratte automaticamente dall'intelligenza artificiale, garantendo così la qualità e la correttezza del dato. Definisce inoltre i requisiti di competenza per i nuovi progetti e analizza i gap formativi.

- **Management / Direzione**: Utente con funzioni prettamente strategiche. Accede al sistema per consultare report aggregati, cruscotti decisionali e dashboard relative ai gap organizzativi, al fine di orientare le assunzioni o i piani di formazione interni.

### 4.2.3 Attori secondari

Gli attori secondari sono entità, spesso non umane, che non avviano direttamente i casi d'uso principali ma forniscono servizi di supporto, elaborazione o fornitura dati necessari affinché il sistema possa completare i *task* richiesti dagli attori primari.

- **LLM (modello Claude Haiku 4.5)**: Rappresenta il motore cognitivo principale per l'analisi avanzata del linguaggio naturale all'interno della pipeline di *parsing*. Integrato tramite API (ad esempio su infrastruttura AWS Bedrock[G]), elabora il testo grezzo dei CV per identificare e strutturare le esperienze e le skill. Svolge il ruolo cruciale di mappare le informazioni estratte su una tassonomia standardizzata (come ESCO), generando il dataset iniziale che verrà poi sottoposto alla necessaria fase di validazione umana da parte del team HR.
- **ESCO Local API**: Costituisce l'infrastruttura tassonomica e semantica di riferimento per la standardizzazione delle competenze. Ospitata internamente per garantire bassissime latenze e totale isolamento dei dati, espone i servizi di ricerca full-text e interrogazione vettoriale necessari per la navigazione del vocabolario europeo. Interviene attivamente sia a supporto dell'utente HR (fornendo funzionalità di autocompletamento durante l'inserimento manuale), sia a supporto della pipeline NLP, agendo come validatore architetturale per allineare, normalizzare e deduplicare le skill grezze estratte dal modello LLM verso un formato informativo unificato.

## 4.3 Casi d'uso

Di seguito vengono esposti i principali raggruppamenti dei casi d'uso del sistema, organizzati seguendo l'architettura per fasi evolutive del progetto. Questa struttura permette di evidenziare le interazioni degli attori con i diversi moduli funzionali e traccia in modo diretto i requisiti obbligatori di sistema.

### 4.3.1 Fase 1: Evoluzione anagrafica competenze

La fase iniziale dello sviluppo ha riguardato l'estensione dell'anagrafica utente preesistente all'interno della piattaforma Nexum, al fine di supportare una gestione strutturata e scalabile delle competenze, abbandonando l'inserimento manuale testuale libero per evitare inconsistenze semantiche.

#### Interrogazione e assegnazione da tassonomia standard

Gli attori possono ricercare le competenze affidandosi al framework ESCO (European Skills, Competences, Qualifications and Occupations). Attraverso l'integrazione di una local API, il sistema interroga questo vocabolario controllato, permettendo all'HR di effettuare l'associazione manuale delle skill all'utente. In questa fase è inoltre possibile configurare una scala di valutazione per definire con precisione i livelli di padronanza della competenza.

#### Creazione skill personalizzate e categorizzazione

Qualora nel database ESCO non sia presente la competenza specifica cercata, il sistema fornisce all'utente HR l'opportunità di definire skill personalizzate (creazione skill), specificandone il nome, la descrizione e la relativa categoria (classificandole in *transversal*, *cross-sector*, *sector-specific*, *occupation-specific*). È inoltre possibile definire relazioni e correlazioni dirette tra competenze diverse all'interno della tassonomia interna.

Diagramma UML[G] dei casi d'uso in figura B.1 nell'Appendice B.

### 4.3.2 Fase 2: Parsing CV e Data Extraction

Per ottimizzare l'esperienza utente e ridurre i tempi operativi della ricerca manuale nel dizionario, è stata implementata una pipeline di automazione basata sul Natural Language Processing (NLP).

#### Upload documenti ed estrazione NLP

Gli utenti possono effettuare l'upload dei propri Curriculum Vitae nei formati standard (PDF, DOC, DOCX). Il sistema, sfruttando un Large Language Model integrato tramite il servizio AWS Bedrock, esegue il parsing del testo per identificare e riconoscere in automatico le skill tecniche o trasversali. Parallelamente, estrae l'esperienza lavorativa e formativa, individuandone gli anni e il contesto di riferimento.

#### Normalizzazione e auto-Assegnazione

I dati grezzi estratti passano per un algoritmo di normalizzazione (mapping) che allinea le entità identificate al dizionario standard ed effettua una deduplicazione per rimuovere eventuali skill ripetute. Il sistema esegue infine un'auto-assegnazione, popolando il profilo dell'utente con i dati normalizzati.

Diagramma UML dei casi d'uso in figura B.2 nell'Appendice B.

### 4.3.3 Fase 3: Profilo dinamico risorsa

Questo modulo è dedicato al mantenimento e alla consultazione del patrimonio di competenze a livello individuale.

#### Visualizzazione e storicizzazione

Il dipendente e l'HR possono visualizzare in un'apposita sezione l'elenco completo delle skill associate all'utente e i relativi livelli. Il sistema rileva e storicizza ogni evoluzione tramite una *timeline delle esperienze*, fornendo una visualizzazione dello storico essenziale per analizzare a posteriori la crescita professionale.

#### Validazione HR

Ogni modifica o inserimento genera un flusso di revisione in cui il team HR procede alla validazione, revisione e conferma definitiva del dato.

Diagramma UML dei casi d'uso in figura B.3 nell'Appendice B.

### 4.3.4 Fase 4: Skill Matching Engine

Questa fase introduce il nucleo analitico del sistema, generando insight diretti a supporto delle decisioni strategiche HR e della crescita del personale.

#### Generazione radar e analisi gap

Il sistema genera automaticamente una rappresentazione grafica a radar per visualizzare le competenze di un utente. Parallelamente, esegue un confronto diretto ruolo-skill, identificando le differenze (gap) tra le competenze idealmente richieste per una mansione e quelle effettivamente possedute dalla risorsa.

#### Clustering dell'aderenza e suggerimenti automatici

Il motore esegue query sul database risorse classificando i candidati in gruppi di idoneità. A fronte dell'analisi, specialmente per le risorse parzialmente idonee, il sistema propone suggerimenti automatici: questi includono percorsi formativi, opportunità di mobilità interna, o l'attivazione di relazioni di apprendimento tramite mentoring aziendale.

Diagramma UML dei casi d'uso in figura B.4 nell'Appendice B.

### 4.3.5 Fase 5: Mappatura progetti e matching

L'ultimo modulo connette in modo proattivo le competenze mappate con le necessità operative o di business dell'azienda.

#### Definizione fabbisogno progettuale e matching risorse

Il Management o l'HR possono inserire a sistema le skill richieste, definendo le competenze necessarie per un nuovo progetto o ruolo. Una volta definiti i criteri, il motore di matching elabora un ranking automatico dei candidati (classificando le risorse più adatte) e procede creando un'associazione diretta tra la risorsa e il progetto.

#### Identificazione dei gap organizzativi

Estendendo l'analisi dal singolo alla collettività, il sistema identifica le carenze organizzative. Aggregando i dati dei profili e i requisiti dei progetti, evidenzia i gap sistemici sulle competenze mancanti, supportando la Direzione nella pianificazione mirata di nuove assunzioni o macro-piani formativi.

Diagramma UML dei casi d'uso in figura B.5 nell'Appendice B.

## 4.4 Requisiti funzionali

I requisiti funzionali definiscono in modo dettagliato le capacità operative e i servizi che il modulo di *Skill Intelligence* deve necessariamente erogare per soddisfare le esigenze espresse dagli attori del sistema. L'intera architettura funzionale è stata segmentata seguendo una classificazione gerarchica delle priorità, distinguendo le funzionalità fondamentali (*Obbligatorie*) da quelle incrementali (*Desiderabili* e *Facoltative*).

Le macro-aree funzionali identificate comprendono:

- **Gestione e Anagrafica delle Competenze:** Corrisponde al nucleo del sistema dedicato alla strutturazione tassonomica delle skill. Include la possibilità per il personale HR di creare, categorizzare (allineandole alle categorie *transversal, cross-sector, sector-specific, occupation-specific*) e correlare tra loro le competenze, nonché di associarle manualmente agli utenti definendo una scala di valutazione standardizzata.

- **Pipeline di Parsing NLP ed Estrazione:** Rappresenta il motore automatizzato del sistema, incaricato di gestire il caricamento dei *Curriculum Vitae* nei formati standard (PDF, DOC, DOCX). Il backend si occupa del parsing del testo libero, del riconoscimento automatico delle entità relative alle competenze e della storicizzazione delle esperienze lavorative e formative. Un sotto-modulo critico gestisce la normalizzazione e la deduplicazione delle skill estratte tramite l'allineamento semantico al dizionario europeo standard ESCO.

- **Profilazione Dinamica e Validazione:** Regola le interazioni e il mantenimento dei dati a livello individuale. Permette la visualizzazione dei profili utente arricchiti da una *timeline* storica delle esperienze, supportando la revisione *human-in-the-loop*[G] operata dall'HR.

- **Motore Algoritmico e Matching:** Governa le funzionalità analitiche di alto livello. Il sistema genera graficamente lo *Skill Radar* del singolo utente e calcola in tempo reale il *gap* rispetto al ruolo aziendale, inserendo l'utente in cluster di idoneità ed emettendo suggerimenti automatici legati a percorsi formativi, mobilità interna o attività di *mentoring*. A livello collettivo, permette di definire il fabbisogno di un progetto e stilare un *ranking* automatico delle risorse più idonee.

L'elenco analitico, formale e codificato di ciascuna singola funzionalità atomica implementata, suddiviso per livello di priorità, è consultabile all'interno delle tabelle riepilogative presenti nell'Appendice B, nello specifico la Tabella B.1 per le componenti di base, la Tabella B.2 per la gestione estesa di ruoli e progetti, e la Tabella B.3 per le interfacce avanzate di monitoraggio.

## 4.5 Requisiti non funzionali

I requisiti non funzionali stabiliscono le caratteristiche qualitative, le restrizioni tecniche e i vincoli infrastrutturali complessivi entro cui il sistema deve operare per garantire la conformità con gli standard ingegneristici della piattaforma Nexum. Tali requisiti sono stati categorizzati in vincoli tecnologici stringenti (*Vincoli*) e metriche misurabili di efficienza (*Qualità*).

L'analisi architetturale ha isolato i seguenti pilastri non funzionali:

- **Vincoli di Integrazione e Stack Tecnologico:** Il modulo sviluppato non dev'essere considerato un'entità software isolata, ma deve integrarsi nativamente con un sistema preesistente. Lo sviluppo ha dovuto rispettare rigorosamente lo stack aziendale, che prevede l'adozione di *Ruby on Rails* in modalità API-only per il backend, del framework *Angular* per il client frontend e di *PostgreSQL* per la persistenza e l'integrità del database relazionale.
- **Infrastruttura Cloud e Modelli AI:** La pipeline di elaborazione semantica e il modulo di *Natural Language Processing* devono appoggiarsi all'infrastruttura di cloud computing *Amazon Web Services* (AWS), sfruttando in modo specifico il servizio gestito *AWS Bedrock* per l'orchestrazione sicura delle chiamate dirette al modello generativo *Claude Haiku 4.5*.
- **Isolamento dei Dati Multi-tenant[G]:** Trattandosi di un'applicazione destinata a organizzazioni multiple, la modellazione logica del database e lo sviluppo delle API devono garantire un isolamento logico dei dati. Le informazioni e i documenti relativi a un'azienda cliente non devono in nessun caso essere accessibili o visibili da utenti appartenenti a tenant differenti.
- **Qualità, Prestazioni e Scalabilità:** Vengono definiti i parametri di efficienza del sistema. Il tempo complessivo richiesto per l'upload, il parsing del testo e la successiva normalizzazione tassonomica di un singolo documento di *Curriculum Vitae* deve attestarsi al di sotto della soglia critica dei dieci secondi. Parallelamente, l'infrastruttura deve essere progettata per scalare orizzontalmente in caso di caricamenti massivi e i ricalcoli delle metriche si *Skill Intelligence* devono avvenire in modalità *near real-time* a seguito di ogni mutazione dello stato del database.

I dettagli formali relativi a tali restrizioni e le metriche prestazionali di riferimento sono esposti in forma tabellare nell'Appendice B, all'interno della Tabella B.4 per quanto

concerne i vincoli architetturali e della Tabella B.5 per i requisiti qualitativi e di prestazione del software.

# Capitolo 5

# Progettazione del sistema

## 5.1 Introduzione

Il presente capitolo ha lo scopo di illustrare l'architettura logica e la progettazione del sistema sviluppato per il modulo "Skill Intelligence". Verranno analizzate le scelte architetturali, i design pattern implementati e le modalità di interazione tra i diversi sotto-componenti software. L'obiettivo è fornire una visione chiara e strutturata di come i requisiti funzionali emersi in fase di analisi siano stati tradotti in una soluzione software robusta e manutenibile, inserita nel contesto tecnologico preesistente della piattaforma aziendale Nexum.

## 5.2 Architettura logica

Il backend dell'applicazione adotta un'architettura a livelli (*Layered Architecture*) rigidamente orientata al pattern *"Skinny Controller, Fat Service"*. Questa scelta strategica è finalizzata a isolare i contesti operativi, garantendo al contempo un'elevata scalabilità orizzontale e una netta separazione delle responsabilità (*Separation of Concerns*).

Il flusso di elaborazione attraversa tre strati logici principali:

- **Livello di Presentazione (Presentation Layer)**: Guidato da controller snelli (*skinny controllers*), il cui unico scopo è la gestione del traffico e del protocollo HTTP. I controller si occupano esclusivamente di intercettare le richieste API, validare i parametri in ingresso, verificare i criteri di sicurezza e i permessi di accesso multi-tenant tramite il modulo di autorizzazione *Pundit*, e impacchettare infine le risposte in formato JSON. Qualsiasi logica computazionale o di calcolo viene rigorosamente delegata allo strato inferiore.

- **Livello di Business Logic (Service Layer)**: Costituisce il centro logico del sistema. È popolato da *Service Objects* isolati, classi a responsabilità singola progettate per eseguire transazioni atomiche e complesse regole di business. Questo livello coordina l'esecuzione di *task* asincroni demandati a processi in background (*Background Jobs*)

e governa le integrazioni con i sistemi esterni. Tra queste, spiccano l'interazione con l'"ESCO Local API" per la gestione della tassonomia ESCO e l'orchestrazione delle chiamate verso i modelli di linguaggio di *AWS Bedrock*, deputati al parsing intelligente e alla riconciliazione dei dati non strutturati estratti dai CV.

- **Livello di Accesso ai Dati (Data Access Layer)**: Gestito mediante i modelli *ActiveRecord* interfacciati con il database relazionale *PostgreSQL*. Questo strato assicura il mantenimento dell'integrità referenziale dei dati, la persistenza dei documenti fisici e degli allegati multimediali tramite il framework *ActiveStorage*, e l'isolamento logico dei dati per ciascuna azienda cliente (*multi-tenant isolation*), centralizzando l'intero patrimonio informativo all'interno di un dizionario ontologico standardizzato.

### 5.2.1 Esempio applicativo: Pipeline di estrazione CV

Per illustrare concretamente l'applicazione del pattern architetturale appena descritto, è possibile analizzare il flusso di esecuzione della pipeline di estrazione e assegnazione automatica delle competenze. Come evidenziato nello schema architetturale del sistema (Figura C.1), il processo attraversa i tre livelli logici in modo sequenziale, isolando le dipendenze. Lo schema è visualizzabile all'Appendice: C.1.

1. **Innesco (Presentation Layer)**: La richiesta di elaborazione del documento viene recepita dal `CvAnalyzersController`, il quale funge unicamente da punto di ingresso. Per evitare di bloccare il thread principale durante l'elaborazione NLP, il controller delega immediatamente l'operazione asincrona a un worker in background, mettendo in coda un `CvAnalysisJob`.

2. **Orchestrazione (Service Layer)**: Il job avvia il servizio principale di orchestrazione (`Process`). Questo Service Object dirige il flusso delegando *sub-task* altamente specifici:

   - `ExtractFromDocument`: incapsula e gestisce la comunicazione con l'API di **AWS Bedrock** per il parsing semantico del CV.
   - `SuggestionMatcher` e `Resolve`: utilizzano classi di ricerca (`Search`, `List`, `Show`) per interrogare la **ESCO Local API**, riconciliando le skill estratte in formato testo libero con la tassonomia europea standard.
   - `TelemetryResolver`: si occupa di aggregare i dati operativi per monitorare l'accuratezza del modello.

3. **Persistenza (Data Access Layer)**: Terminata la fase di normalizzazione logica, il servizio `Process` dialoga direttamente con i modelli ActiveRecord per il consolidamento dei dati. Vengono creati o aggiornati i record relativi a `CvAnalysis`, `User` e `Company`; le competenze confermate popolano la tabella `Standard Skill` e,

infine, il sistema ufficializza l'assegnazione automatica generando il vincolo relazionale in `UserSkill`. L'esito e le metriche dell'elaborazione vengono storicizzati in `SkillExtractionLog`.

## 5.3 Architettura frontend (Client)

La logica dell'interfaccia utente è interamente strutturata attorno al pattern architetturale *Model-View-ViewModel* (MVVM[G]), un paradigma che si integra in modo nativo con l'architettura del framework Angular. Rispetto al classico schema MVC, l'MVVM permette di disaccoppiare drasticamente la rappresentazione visiva dalla logica di business transazionale, facilitando la manutenibilità del codice e l'estendibilità delle interfacce.

Il sistema suddivide le proprie responsabilità strutturali in tre livelli ben definiti:

- **View (La Vista)**: Rappresentata dai file di template `.html`[G]. In Angular, la View non è un elemento passivo, ma interagisce attivamente con l'applicazione grazie al meccanismo del *data-binding*. Il suo compito è limitato alla presentazione visiva dei dati e alla cattura degli eventi utente.

- **ViewModel (Il Modello della Vista)**: Costituito dalle classi TypeScript dei singoli componenti. Il ViewModel funge da intermediario: non manipola direttamente i dati grezzi provenienti dal database, ma modella e mantiene uno stato applicativo ottimizzato specificamente per le esigenze della View, gestendo lo stato transitorio della UI[G] e le interazioni dell'utente.

- **Model (Il Modello)**: Rappresenta il nucleo dei dati e della logica di business lato client. Questo strato è implementato attraverso i *Services* iniettabili di Angular e dalle interfacce TypeScript che definiscono rigorosamente la forma dei dati contrattati con il backend. I servizi si occupano di effettuare le richieste HTTP asincrone e restituire flussi informativi puri al ViewModel.

L'architettura del frontend estende inoltre l'approccio MVVM classico introducendo i principi della programmazione reattiva funzionale (*Functional Reactive Programming*). Sfruttando la libreria RxJS e la gestione nativa degli *Observable*, lo stato del ViewModel evolve come un flusso continuo di dati guidato dagli eventi (*event-driven*), garantendo una sincronizzazione in tempo reale.

Il livello di visualizzazione è organizzato in un albero gerarchico di componenti, ciascuno responsabile di una specifica porzione visiva e della relativa logica di interazione (ad esempio, il componente per la visualizzazione del radar o quello per l'upload del CV). La comunicazione tra il client Angular e il server avviene in modo asincrono attraverso chiamate HTTP, aderendo pienamente allo stile architetturale **REST**[G] (*Representational State Transfer*). Il Frontend interroga le API esposte per inviare file, richiedere elaborazioni e consumare i dati (ricevuti in formato JSON) necessari al rendering delle dashboard.

## 5.4 Design pattern

Il Backend è sviluppato in **Ruby on Rails** e funge da motore centrale per l'elaborazione NLP, l'autenticazione, la validazione e la persistenza dei dati. L'architettura interna sfrutta i pattern standard promossi dal framework, adattati per un'infrastruttura *API-only*:

- **Model-View-Controller (MVC)**: È il pattern architetturale fondante su cui si basa Rails. Nel contesto di questo progetto, la componente "View" tradizionale (responsabile della generazione di HTML) viene sostituita da serializzatori che trasformano gli oggetti di dominio in risposte JSON strutturate. I "Controller" gestiscono il routing delle richieste API in ingresso e mantengono un profilo "sottile" (*thin controller*), limitandosi a delegare l'esecuzione ai Model o ai Service, mentre i "Model" incapsulano le regole di business.
- **Active Record**: Pattern per la gestione dell'Object-Relational Mapping (ORM). Consente alle entità del dominio (come le Risorse o i Progetti) di interfacciarsi in modo trasparente e orientato agli oggetti con il database.
- **Service Object Pattern**: Per evitare di sovraccaricare i Model con logiche di integrazione di terze parti, è stato adottato il pattern dei *Service Objects*. Si tratta di classi Ruby pure (PORO - Plain Old Ruby Object) incaricate di svolgere una singola operazione di business complessa. Nello Skill Radar, un Service Object dedicato si occupa di orchestrare l'interazione con l'intelligenza artificiale: impacchetta il testo del CV elaborato, applica le credenziali e le policy necessarie, e gestisce in isolamento la chiamata esterna alle API di **AWS Bedrock**, restituendo l'output NLP strutturato al resto dell'applicazione.

## 5.5 Gestione della tassonomia e persistenza dei dati

Un aspetto fondamentale dell'architettura logica riguarda la gestione del dizionario delle competenze (standard ESCO), progettata per ottimizzare la velocità di matching e l'integrità relazionale:

1. **ESCO Local API e Motore di Ricerca**: Data la vastità della tassonomia ESCO, la ricerca a runtime delle competenze non grava sul database transazionale. Viene invece utilizzata una *Local API* supportata da un motore di ricerca dedicato (ottimizzato tramite un database vettoriale per la ricerca semantica). Questo strato permette di garantire risultati rapidi in risposta agli input testuali del frontend o durante il processo di mapping eseguito dall'LLM.
2. **Database Relazionale (PostgreSQL)**: Nel momento in cui una competenza estratta o ricercata viene definitivamente validata e collegata a un'entità del sistema (come la referenza a un utente o il requisito di un progetto), essa abbandona il solo contesto

vettoriale e viene persistita permanentemente all'interno del database relazionale principale in **PostgreSQL**. Questo doppio passaggio assicura il mantenimento della rigorosa integrità referenziale necessaria all'infrastruttura Nexum, garantendo transazioni sicure e query analitiche affidabili tramite Active Record.

Per supportare questa architettura ibrida e garantire la coerenza dei dati, lo schema logico relazionale (il cui diagramma Entità-Relazione completo è riportato in Appendice) si articola attorno a entità cardine fortemente normalizzate:

- `StandardSkill`: Rappresenta il fulcro dell'intero sistema. Agisce come nodo centrale a cui convergono i requisiti di ruolo, le abilità storiche degli utenti e i percorsi formativi, venendo adottata formalmente dall'azienda (`Company`) tramite l'associazione *Adoption*.
- `User` e `UserSkill`: Gestiscono l'anagrafica dei dipendenti impiegati nell'azienda (*Employment*) e la relativa mappatura delle competenze (*Proficiency*) correntemente possedute.
- `RoleProfile` e `RoleProfileSkill`: Definiscono i modelli di ruolo ideali (*Definition*) previsti dall'organigramma. L'entità debole `RoleProfileSkill` funge da associazione pivot, esprimendo i requisiti (*Requirement*) specifici di competenza e i relativi livelli attesi per ciascun ruolo.
- `Project` e `TeamMember`: Orchestrano l'allocazione delle risorse sui progetti aziendali (*Ownership* e *Allocation*). Tracciano l'effettiva partecipazione (*Participation*) degli utenti e la loro eventuale assegnazione formale a specifici `RoleProfile` all'interno del team.
- `Experience` e `ExperienceSkill`: Ricostruiscono analiticamente il *background* lavorativo dell'utente, legando (*Application*) ogni esperienza lavorativa pregressa alle competenze specifiche sviluppate durante quel lasso di tempo.
- `TrainingAssignments` e `TutoringAssignments`: Supportano i processi di *upskilling* continuo. Tracciano rispettivamente l'iscrizione autonoma a percorsi formativi (*Enrollment*) volti all'acquisizione di skill *target*, e le relazioni strutturate da utente a utente (*Mentorship* e *Apprenticeship*) focalizzate sul trasferimento di una particolare competenza.
- `CvAnalysis` e `SkillExtractionLog`: Sovraintendono ai processi NLP. Modellano l'analisi temporanea per la valutazione documentale dell'utente (*Evaluation*) e mantengono uno storico (*Logging*) delle associazioni algoritmiche effettuate per favorire il *Machine Learning* euristico del sistema di risoluzione delle ambiguità.

Lo schema ER del database descritto è visualizzabile nell'Appendice C.2.

## 5.6 Considerazioni sul trattamento dei dati e conformità GDPR

Il trattamento dei dati personali all'interno di un modulo HR basato su tecniche di *Natural Language Processing* richiede un'attenta analisi ingegneristica del ciclo di vita delle informazioni, in conformità con le normative vigenti in materia di privacy (GDPR[G]). All'interno dell'infrastruttura sviluppata, la pipeline di gestione documentale è stata ingegnerizzata per ridurre al minimo la persistenza di dati intermedi non strutturati. Il flusso prevede che, al momento del caricamento del *Curriculum Vitae* da parte dell'utente o del personale HR, il file in formato grezzo (PDF, DOC, DOCX) venga inviato direttamente e senza alterazioni preliminari all'infrastruttura cloud. La persistenza a lungo termine del documento è demandata al servizio di *object storage* Amazon S3. Per ogni risorsa censita a sistema viene memorizzato un singolo file univoco all'interno di un *bucket* dedicato. L'isolamento logico e la sicurezza dei dati sensibili vengono garantiti a livello applicativo attraverso la struttura *multi-tenant* nativa della piattaforma Nexum: l'accesso al file memorizzato su S3 e ai metadati relazionali associati è rigidamente vincolato all'identificativo dell'organizzazione di appartenenza (*company_id*), impedendo qualsiasi forma di accesso trasversale non autorizzato.

## 5.7 Prompt Engineering e strutturazione dell'output JSON

Il motore cognitivo del sistema di estrazione NLP si basa sull'integrazione del modello *Claude Haiku 4.5* tramite le API di *AWS Bedrock*. Trattandosi di un modello di linguaggio generativo, una delle sfide ingegneristiche principali ha riguardato la determinazione di un *System Prompt*[G] deterministico. L'obiettivo è duplice: mitigare i fenomeni di allucinazione intrinseci ai modelli probabilistici e istruire l'algoritmo a restituire un *output* rigidamente strutturato, direttamente mappabile sulle entità del database relazionale senza richiedere complessi passaggi intermedi di *parsing* testuale.

Il *System Prompt* è stato ingegnerizzato adottando una combinazione di tecniche di *Role Prompting* e *Constraint Enforcement*. Il modello viene inizialmente isolato in un contesto operativo ben definito, istruendolo ad agire esclusivamente come un sistema avanzato di *Named Entity Recognition* specializzato nell'analisi delle risorse umane.

Per ottimizzare l'estrazione e garantire l'integrità del dato prima del passaggio al motore di ricerca semantico, sono state codificate nel prompt le seguenti euristiche e regole logiche:

- **Estrazione multidimensionale:** Il modello è istruito a scansionare il testo libero del *Curriculum Vitae* per identificare non solo le competenze strettamente tecniche e metodologiche (*hard skills*), ma anche quelle trasversali (*soft skills*), inferendole analiticamente dal contesto e sintetizzandole in brevi frasi esplicative.
- **Localizzazione e traduzione dinamica:** Sfruttando le funzionalità di interpolazione delle stringhe lato backend, il prompt integra la variabile applicativa `#{@language}`.

Questo vincola il modello a tradurre e normalizzare autonomamente ogni competenza rilevata nella lingua corrente dell'interfaccia utente selezionata, garantendo l'omogeneità linguistica del *dataset*.

- **Normalizzazione sintattica:** Per evitare la proliferazione di varianti lessicali della medesima competenza, viene imposta una regola di semplificazione grammaticale. Il modello deve isolare il nucleo tecnologico o metodologico eliminando i sintagmi verbali o preposizionali complessi (ad esempio, trasformando la locuzione "programmazione in Python" nel token atomico "Python").
- **Raffinamento ontologico (Regola ESCO):** Al fine di predisporre il dato a un'efficace riconciliazione con la tassonomia europea ESCO, viene introdotto un vincolo di specificità domain-specific. Qualora il documento presenti termini eccessivamente generici (come il lemma "Sviluppo"), il modello ha l'obbligo di analizzare il contesto circostante per sostituirlo con l'espressione specialistica di riferimento (ad esempio, "Sviluppo web").
- **Gestione deterministica del fallback:** Per minimizzare l'occorrenza di risposte spurie o descrittive in caso di documenti non idonei o privi di informazioni pertinenti, viene definita una sanzione logica: l'obbligo di restituire un array vuoto impostando il contatore dei metadati a zero.

Un aspetto critico della progettazione ha riguardato il totale divieto di inserire spiegazioni, saluti o testo discorsivo prima o dopo la struttura dati, inclusa l'esplicita proibizione dell'utilizzo della formattazione *Markdown* standard (ovvero i tre *backtick* "`` `json ``[G]). Questa restrizione garantisce che il *payload* ricevuto dall'interfaccia di AWS Bedrock sia un oggetto JSON puro, immediatamente serializzabile dalle classi *Service* del backend.

Lo schema sintattico richiesto al modello rispetta rigorosamente la seguente struttura:

```
{
  "total_skills": <numero_intero>,
  "skills": [
    {
      "name": "<nome_della_skill_normalizzato>"
    }
  ]
}
```

Grazie a questo approccio, il sistema beneficia di una drastica riduzione della latenza di elaborazione post-estrazione, delegando la logica di formattazione direttamente allo strato di inferenza del Large Language Model.

## 5.8 Modellazione logica e visualizzazione dei grafi

Le principali strutture visuali implementate nell'interfaccia per l'analisi del profilo individuale e la relativa mappatura tassonomica includono:

- **Sunburst Chart[G] (Mappa Competenze):** Un grafico radiale *space-filling* utilizzato per rappresentare gerarchicamente la tassonomia delle competenze associate alla risorsa. L'anello interno suddivide il profilo in macro-categorie strutturali (ad esempio, distinguendo tra competenze *transversal*, *sector-specific*, *cross-sector* e *occupation-specific*), mentre gli anelli esterni si espandono in modo concentrico mostrando le specifiche sotto-competenze tecniche, trasversali o gli strumenti tecnologici (come linguaggi di programmazione e database). Questa mappatura permette di cogliere visivamente e in modo immediato la densità, la varietà e la distribuzione del bagaglio di competenze della singola risorsa [32].
- **Timeline[G] Verticale (Timeline Esperienze):** Un componente cronologico strutturato a grafo lineare che traccia il percorso storico del candidato. Ogni nodo rappresenta una specifica esperienza acquisita nel tempo (lavorativa o accademica, differenziata visivamente tramite icone dedicate come la valigetta o l'abitazione). Questa vista fornisce il contesto temporale essenziale per l'analisi, permettendo di correlare la maturazione delle competenze presenti nella mappa radiale con l'effettiva evoluzione della carriera o della formazione dell'utente.

L'integrazione affiancata di queste due viste all'interno della medesima dashboard fornisce una panoramica completa e complementare: mentre la Mappa quantifica e categorizza il capitale intellettuale secondo una struttura ad albero, la Timeline ne contestualizza l'acquisizione sull'asse temporale. Dato che le informazioni vengono estratte ed elaborate dalla pipeline NLP lato backend, i componenti dell'interfaccia popolano dinamicamente la topologia dei grafici ad ogni aggiornamento o nuova elaborazione dei documenti.

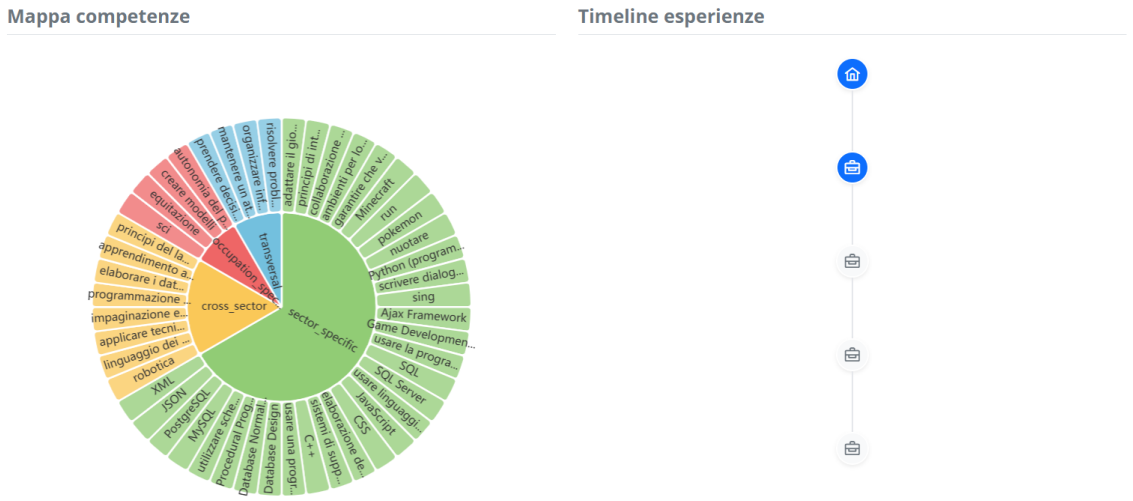


Figura 5.1: Grafici di analisi del profilo utente: Sunburst Chart e Timeline

### 1. Grafo Dipendente-Skill (Mappatura delle Competenze)

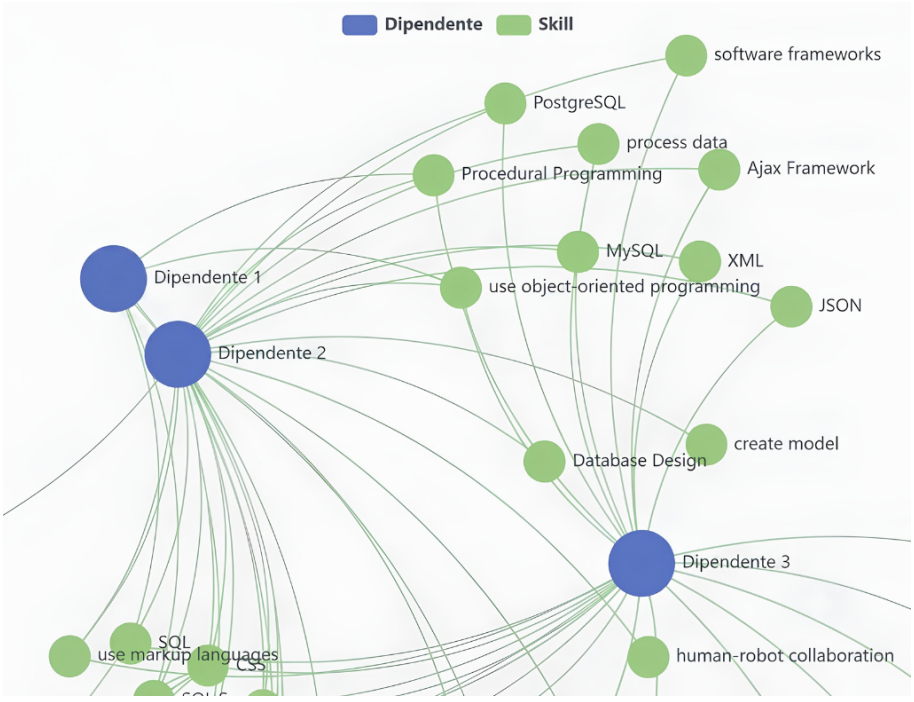


Figura 5.2: Grafo bipartito delle competenze dei dipendenti aziendali.

**Spiegazione:** Visualizza l'assegnazione diretta delle conoscenze all'interno dell'organico. I nodi sono divisi in due insiemi: il capitale umano (nodi blu, Dipendenti) e la tassonomia delle conoscenze (nodi verdi, Skill).

**Significato degli archi:** Un arco indica il *possesso* di una specifica competenza da

parte di un dipendente, validata a sistema tramite parsing del CV o inserimento manuale.

**Utilizzo in analisi:** Strumento fondamentale per l'*Intelligence* HR. Permette di identificare visivamente raggruppamenti (cluster) di competenze tecnologiche, individuare i "colli di bottiglia" (skill critiche possedute da un solo utente) e identificare rapidamente i dipendenti più versatili (nodi hub con un elevato numero di archi in uscita).

### 2. Grafo Ruoli-Skill (Definizione dei Fabbisogni)

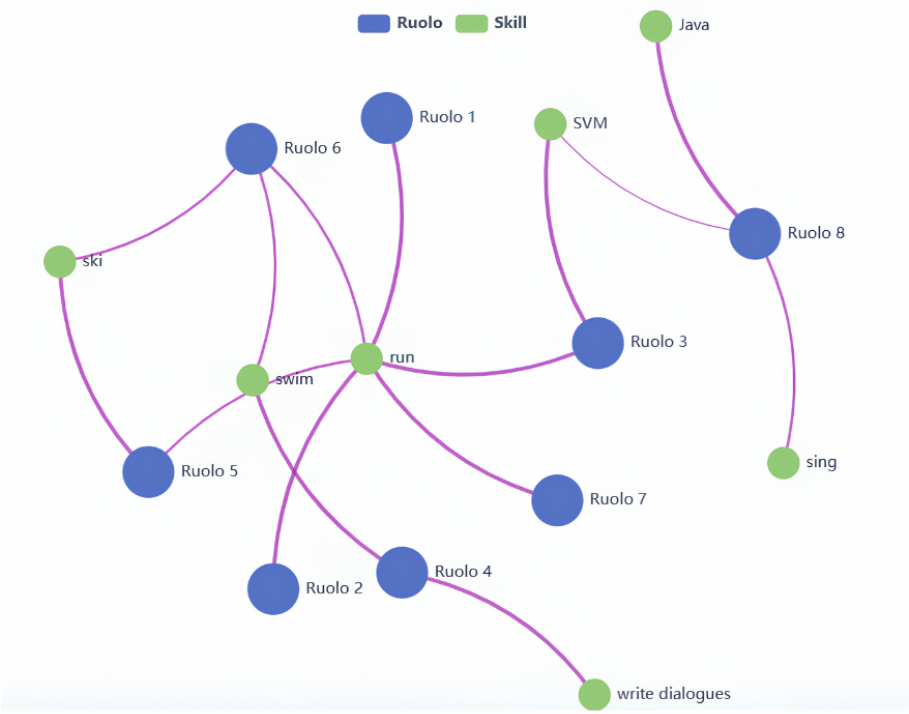


Figura 5.3: Mappatura delle competenze necessarie per ricoprire gli specifici ruoli aziendali.

**Spiegazione:** Illustra l'architettura dei profili professionali aziendali (*Role Profiles*), collegando le mansioni (nodi blu, Ruolo) alle competenze richieste per svolgerle adeguatamente (nodi verdi, Skill).

**Significato degli archi:** Un arco rappresenta un *requisito formale*. Indica che una determinata skill è considerata fondamentale per l'esercizio di un ruolo specifico.

**Utilizzo in analisi:** Supporta la direzione nello studio della trasversalità aziendale. Se un nodo Skill (es. "run" in Figura 5.3) è collegato a numerosi ruoli, significa che si tratta di una competenza nucleare (*core competence*) per l'azienda. Facilita inoltre la pianificazione della mobilità interna: ruoli che condividono molti archi verso le stesse skill indicano passaggi di carriera facilmente attuabili (basso *skill gap*$^G$).

### 3. Grafo delle Collaborazioni (Dinamiche Sociali)

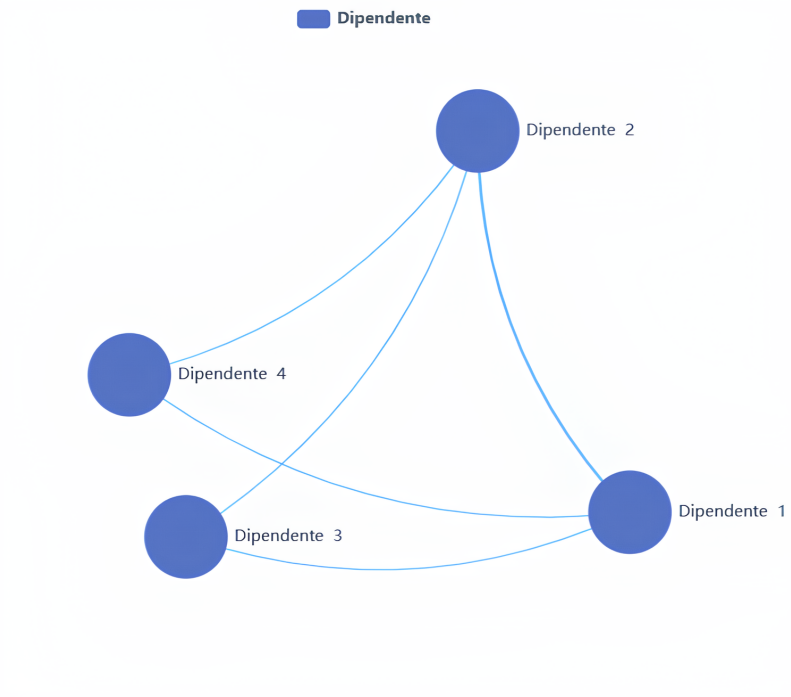


Figura 5.4: Rete di interazioni e legami operativi tra i dipendenti.

**Spiegazione:** È una rete omogenea composta esclusivamente da nodi di tipo "Dipendente", mirata a visualizzare i rapporti lavorativi all'interno dell'organizzazione.

**Significato degli archi:** L'arco rappresenta un *legame di collaborazione attiva* o un'interazione pregressa tra due risorse (ad esempio, l'aver lavorato nello stesso team di progetto o l'essere legati da un rapporto di *mentoring*).

**Utilizzo in analisi:** È la base per l'*Organizational Network Analysis* (ONA[G]). Aiuta il management a scoprire le reali dinamiche sociali, che spesso differiscono dall'organigramma formale. Permette di individuare i dipendenti chiave che fungono da ponte tra diversi dipartimenti (promuovendo la condivisione della conoscenza) o, al contrario, di rilevare risorse/team eccessivamente isolati per i quali pianificare attività di *team building*.

### 4. Grafo dell'Ecosistema di Progetto (Allocazione Risorse)

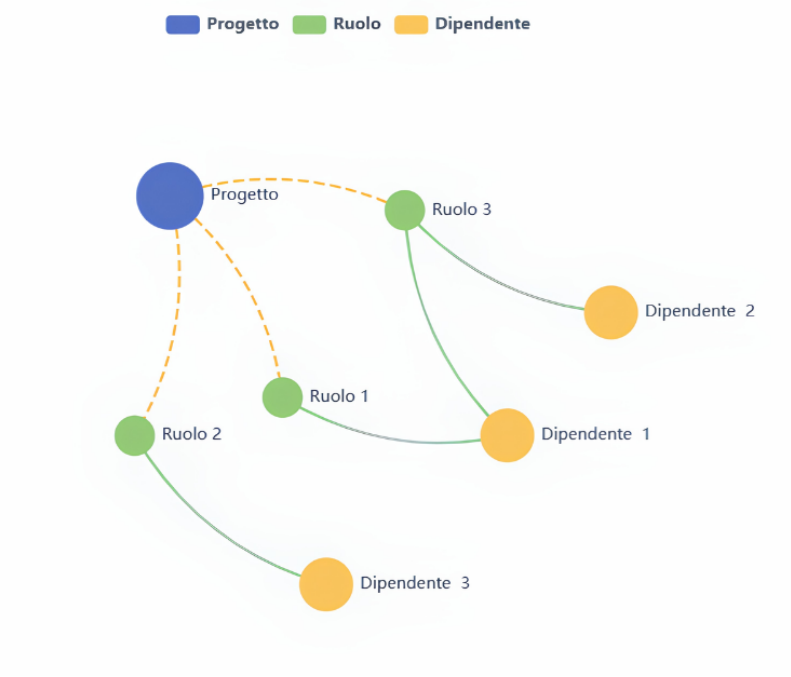


Figura 5.5: Vista gerarchica della composizione, del fabbisogno e dell'allocazione su un progetto.

**Spiegazione:** Fornisce una panoramica globale e multilivello dello stato di un progetto, mettendo in relazione tre diverse entità: il Progetto stesso (nodo blu), i Ruoli richiesti per portarlo a termine (nodi verdi) e i Dipendenti effettivamente assegnati (nodi gialli). **Significato degli archi:** Il grafo adotta la seguente differenziazione semantica:

- *Archi tratteggiati (Progetto → Ruolo):* Rappresentano il *fabbisogno* o la pianificazione, indicando che il progetto necessita di quella specifica figura professionale per essere completato.
- *Archi continui (Ruolo → Dipendente):* Rappresentano l'*allocazione effettiva*, confermando che una specifica risorsa umana è stata assegnata a coprire quel ruolo all'interno del progetto.

**Utilizzo in analisi:** Strumento fondamentale per i Project Manager. Evidenzia a livello visivo se un progetto presenta criticità operative: la presenza di un nodo "Ruolo" privo di archi continui verso un "Dipendente" segnala immediatamente una posizione scoperta (*vacancy*) che blocca l'avanzamento dei lavori, suggerendo l'attivazione immediata del motore di matching descritto nella Fase 5.

## 5.9 Piano di sviluppo

### 5.9.1 Metodologia di implementazione

Lo sviluppo del modulo è stato condotto adottando un approccio incrementale e stratificato, procedendo per iterazioni successive dai livelli di persistenza dei dati fino ai livelli di presentazione visiva. Ogni nuova funzionalità è stata integrata partendo dalla definizione dello schema relazionale tramite le migrazioni del framework, garantendo fin da subito i vincoli di integrità sul database.

Successivamente, si è proceduto alla stesura della logica di dominio, incapsulata all'interno di classi a singola responsabilità, le quali espongono i servizi tramite interfacce API RESTful. A valle della validazione di tali servizi, lo sviluppo si è spostato sul livello client, dove i dati vengono consumati, elaborati e presentati all'utente finale. Questa rigida separazione architetturale ha permesso di collaudare ogni strato in modo isolato, assicurando l'affidabilità delle singole componenti prima della loro integrazione finale.

### 5.9.2 Scomposizione architetturale del contributo implementativo

Al fine di delineare con precisione la portata dello sviluppo effettuato rispetto alle componenti preesistenti della piattaforma ospitante, si riporta di seguito la scomposizione degli artefatti software realizzati. Il contributo ingegneristico si è concentrato sulla progettazione e sull'implementazione dei seguenti livelli:

- **Livello Dati e Logica di Dominio (Ruby on Rails):** L'ecosistema ospitante forniva la gestione dell'autenticazione e i modelli anagrafici primari (`User`, `Company`). Lo sviluppo del modulo ha pertanto richiesto la modellazione relazionale e la stesura delle migrazioni strutturali necessarie all'introduzione di diverse nuove entità, tra le quali: `StandardSkill` per la creazione del dizionario interno; `UserSkill`, `Experience` e `ExperienceSkill`, destinate all'estensione strutturata dei profili professionali dei dipendenti; `CvAnalysis` e `SkillExtractionLog`, funzionali all'orchestrazione e al tracciamento del parsing documentale tramite LLM; nonché `RoleProfile` e `RoleProfileSkill`, preposte alla formalizzazione dei modelli di ruolo e dei fabbisogni aziendali. La logica applicativa è stata infine incapsulata all'interno di classi di servizio dedicate (quali, ad esempio, `CvAnalyzer::Process` e `Candidate::List`), le quali implementano in modo isolato gli algoritmi di business e le funzionalità precedentemente descritte.

- **Integrazione Cloud e Pipeline NLP (AWS):** Avvalendosi dell'infrastruttura di rete preconfigurata, lo sviluppo si è concentrato sull'integrazione programmatica del motore semantico. Ciò ha comportato la stesura della classe di servizio `CvAnalyzer::ExtractFromDocument`, specificamente incaricata di gestire la comunicazione a basso livello con le API di AWS Bedrock. Questo modulo si occupa della codifica del documento in formato Base64, dell'invocazione del modello generativo

e, soprattutto, della sanitizzazione dell'output tramite espressioni regolari (*regex*) per isolare esclusivamente il *payload* JSON valido. A questo strato si affiancano l'ingegnerizzazione testuale del *prompt* di sistema, volta a imporre comportamenti deterministici al modello, e l'implementazione dei processi asincroni (`CvAnalysisJob`) necessari a delegare l'elaborazione documentale senza introdurre latenze nel ciclo di esecuzione principale del server.

- **Livello Client e Visualizzazione Dati (Angular):** A fronte di una struttura di base dell'applicativo fornita dal sistema (autenticazione, navigazione globale e librerie di componenti standard), il contributo ha interessato lo sviluppo delle logiche di presentazione del modulo HR. Sono stati implementati i servizi per la gestione dello stato e per la comunicazione HTTP con il server, le interfacce per la validazione dei dati e la logica dei componenti in TypeScript.

Per poter visualizzare un esempio semplificato dell'implementazione della classe che consente la comunicazione con il modello generativo, si rimanda all'appendice C.3.

# Capitolo 6

# Verifica e validazione

## 6.1 Introduzione

Il presente capitolo illustra le strategie di collaudo, le metodologie di testing e i processi di validazione adottati per garantire la robustezza, la sicurezza e l'affidabilità del modulo *Skill Intelligence* all'interno della piattaforma Nexum. Trattandosi di un sistema aziendale che gestisce dati sensibili (HR) e orchestra logiche di business complesse mediate dall'Intelligenza Artificiale, la fase di verifica ha assunto un ruolo centrale nel ciclo di vita dello sviluppo del software.

## 6.2 Strategia e metodologia di testing

Per rispondere alla complessità dell'architettura a servizi (Service-Oriented) di Nexum, la metodologia di validazione si è basata sul modello della *Test Pyramid* (Piramide dei Test). Questo approccio prevede una base ampia e solida di test di unità, veloci e isolati, sormontata da un livello intermedio di test di integrazione per verificare la comunicazione tra i moduli, fino ad arrivare ai test end-to-end (E2E[G]) per la validazione dei flussi utente.

In linea con la netta separazione delle responsabilità tra client e server, i flussi di test sono stati rigidamente divisi:

- **Backend Testing:** focalizzato sull'integrità dei dati, sull'efficienza degli algoritmi, sulla sicurezza degli endpoint e sul mocking[G] delle dipendenze di rete.
- **Frontend Testing:** incentrato sulla reattività dell'interfaccia, sulla corretta renderizzazione dei componenti grafici (es. grafi ECharts[G]) e sulla gestione dello stato applicativo.

## 6.3 Test del backend

Nel progetto Nexum è stata implementata un'architettura a livelli (layered architecture) fortemente orientata al pattern *Skinny Controller, Fat Service*, pensata per garantire

massima scalabilità e isolamento delle responsabilità. Questa scelta architetturale ha guidato direttamente la strategia di testing lato server, permettendo di collaudare la logica di business in totale isolamento rispetto al livello di trasporto HTTP.

### 6.3.1 Test di unità (Unit Testing)

I test di unità costituiscono una parte fondamentale della validazione del backend. Essi si concentrano su due componenti fondamentali dell'architettura:

- **Modelli (ActiveRecord):** I test verificano l'integrità referenziale su PostgreSQL, le validazioni stringenti sui campi, la gestione del ciclo di vita degli allegati (tramite *ActiveStorage*) e l'isolamento multi-tenant, garantendo che i dati di un'azienda cliente non siano mai accessibili da un'altra.
- **Service Objects (Business Logic):** Il nucleo logico dell'applicazione risiede in queste classi isolate e mono-responsabilità. I test di unità su questi servizi verificano la correttezza degli algoritmi, come il partizionamento dei candidati in *bucket* (perfect match, partial match, unsuitable) in base al calcolo del gap di competenze, o l'accuratezza del *TelemetryResolver* nell'assegnazione automatica basata sullo storico aziendale. I test garantiscono inoltre l'esecuzione sicura delle transazioni sul database, verificando i meccanismi di *rollback* in caso di fallimento parziale.

### 6.3.2 Test di integrazione e API

Il livello di presentazione è affidato a Controller snelli che fungono unicamente da direttori del traffico HTTP. I test di integrazione a questo livello si focalizzano sulla verifica della corretta interconnessione del sistema:

- **Routing e Sicurezza:** Viene testata la corretta risoluzione delle rotte e la reiezione delle richieste non autorizzate. Si verifica che le policy di autorizzazione (*Pundit*) impediscano l'accesso orizzontale o verticale non consentito (es. un dipendente che tenta di eliminare una skill globale).
- **Contratti JSON:** Si valuta che i payload di richiesta vengano correttamente parsati e che le risposte JSON emesse dai Controller rispettino rigorosamente i contratti pattuiti con il frontend, includendo la corretta serializzazione dei dati aggregati dai Service.

### 6.3.3 Mocking e test dei servizi esterni

Una sfida tecnica rilevante nel testing del backend di Nexum è stata la presenza di dipendenze esterne critiche: l'API semantica del catalogo europeo ESCO e l'intelligenza artificiale di AWS Bedrock, utilizzata per il parsing e l'estrazione delle entità dai CV. Effettuare richieste di rete reali durante i test unitari avrebbe reso la suite fragile, lenta e

soggetta a limiti di *rate-limiting* imposti dai provider. Per ovviare a questo problema, è stata adottata una rigorosa strategia di *Mocking*[G] e *Stubbing*[G]:

- **Intercettazione di rete (WebMock):** Per le integrazioni HTTP il traffico in uscita è stato bloccato e sostituito con risposte simulate (stub) riproducenti fedelmente le strutture dati complesse in formato HAL+JSON restituite da ESCO. Questo ha permesso di testare la logica di estrazione e normalizzazione del dizionario ontologico in modo deterministico.
- **Mocking dei Client SDK:** Per il servizio `ExtractFromDocument`, le istanze del client *AWS::BedrockRuntime* sono state isolate mediante "mock objects" in memoria. In questo modo si è potuto verificare non solo la corretta gestione dei payload e delle eccezioni IA, ma anche l'esatta composizione del *System Prompt* generato dinamicamente, assicurandosi che le istruzioni per la prevenzione dei duplicati venissero correttamente iniettate senza invocare il modello linguistico reale.

## 6.4 Test del frontend

La validazione del livello di presentazione è fondamentale per garantire l'affidabilità della Single Page Application in Angular. I test si sono concentrati sull'isolamento dei moduli, la corretta interazione con le API di backend e la reattività dell'interfaccia.

Per i test unitari è stato adottato **Jest**, scelto per le prestazioni superiori del suo ambiente simulato rispetto a soluzioni tradizionali. L'infrastruttura si basa sulla classe `TestBed` di Angular, essenziale per configurare dinamicamente dipendenze fittizie e collaudare i componenti in un ambiente isolato e controllato.

La sfida architetturale principale è stata la gestione delle dipendenze da librerie esterne, come i moduli di shell o gli strumenti di *data visualization*. Per garantire la natura puramente unitaria dei test e prevenire errori legati alla risoluzione delle dipendenze, è stato implementato un *mocking*[G] sistematico tramite `jest.mock()`. Questo approccio ha permesso di iniettare costrutti simulati per riprodurre il comportamento atteso delle librerie esterne, senza introdurre *side-effect* nel ciclo di test.

### 6.4.1 Test dei componenti e della vista

Il collaudo dei componenti ha avuto come obiettivo principale la verifica della logica di presentazione e dell'integrazione tra il template HTML e il controller TypeScript. Utilizzando `ComponentFixture`, è stato possibile simulare il ciclo di vita dei componenti (come `ngOnInit`), intercettare i cambiamenti di stato e ispezionare le reazioni del DOM[G].

Un focus particolare è stato dedicato ai componenti ad alta interattività, come le interfacce per la gestione dei progetti (`ProjectFormComponent`) e per l'analisi dei candidati (`RoleProfilesCandidatesComponent`). I test implementati coprono diversi aspetti critici:

- **Gestione dei Form Reattivi**: È stata collaudata la validazione dei moduli (*Reactive Forms*), assicurandosi che operazioni critiche (come il salvataggio di un progetto) venissero bloccate o permesse in base allo stato di validità dei campi obbligatori.
- **Interazioni asincrone e Race Conditions**: I componenti frontend gestiscono complesse interazioni con l'utente (es. aggiunta e rimozione massiva di ruoli da un progetto). I test hanno validato il corretto funzionamento dell'approccio *Single Source of Truth*[G]: si è verificato che le operazioni asincrone, mediate da librerie esterne (come i dialoghi modali o gli spinner di caricamento), ricaricassero lo stato dal server solo al termine effettivo della transazione, prevenendo sdoppiamenti dell'interfaccia o la comparsa di dati incoerenti.
- **Mocking dei servizi UI**: Per testare i componenti in totale isolamento, servizi ausiliari per la notifica utente (`ToastrService`), per il caricamento (`SpinnerService`) e per la traduzione dinamica (`TranslateService`) sono stati sostituiti con spie controllate tramite `jest.fn()`. Questo ha permesso di asserire (`toHaveBeenCalledWith`) l'esatta invocazione dei messaggi di successo o di errore simulando, ad esempio, fallimenti fittizi nelle chiamate HTTP.

### 6.4.2 Test dei servizi e gestione dello stato

Se i componenti gestiscono la presentazione, i Servizi Angular incapsulano la logica di business e la comunicazione con le API REST[G] del backend. Il testing di questo strato è stato realizzato sfruttando l'`HttpClientTestingModule` e l'`HttpTestingController`, strumenti nativi di Angular che permettono di bypassare la rete reale, intercettare le richieste HTTP in uscita e simulare le risposte del server.

L'implementazione dei test sui servizi si è articolata attorno a criteri rigorosi di validazione:

- **Generazione dinamica degli Endpoint**: Poiché l'architettura del sistema è multi-tenant, le URL[G] delle API variano in base al contesto aziendale corrente (`companyId`). I test hanno validato che ogni servizio costruisse le URI[G] corrette iniettando dinamicamente identificativi fittizi (tramite il mock del `CompanyDataService`).
- **Validazione dei metodi e dei Payload**: Per ogni operazione CRUD[G] (Creazione, Lettura, Aggiornamento, Eliminazione), il controller di test ha verificato che il metodo HTTP utilizzato fosse quello semanticamente corretto (GET, POST, PATCH, DELETE). Inoltre, è stata impiegata la funzione `expectOne` per intercettare la richiesta e asserire (`toEqual`) che il corpo della stessa (*payload*) contenesse i dati formattati correttamente.
- **Mapping dei dati**: Diversi servizi (come lo `UserSkillService`) si occupano di mappare valori semantici dell'interfaccia in valori numerici per il database (es. convertendo livelli di competenza testuali in interi, oppure estraendo UUID da URI

completi). I test unitari hanno verificato la correttezza di queste trasformazioni prima che i dati venissero affidati al modulo HTTP.

- **Gestione dei Query Parameters**: Un ultimo elemento critico validato riguarda il passaggio dinamico dei parametri via querystring (come il parametro `language` per l'internazionalizzazione lato server). È stato testato l'utilizzo corretto della classe `HttpParams`, garantendo che il frontend non formattasse URL errate o parametri indefiniti (`undefined`) nel caso di traduzioni mancanti.

Attraverso questo strato di test sui servizi, si è garantita la stabilità del contratto dati tra il client Angular e il server backend, isolando preventivamente errori di routing o disallineamenti nelle strutture JSON.

## 6.5 Collaudo ed End-to-End (User Acceptance Testing)

A completamento della *Test Pyramid*, la validazione del sistema non si è limitata all'esecuzione automatizzata dei test di unità e di integrazione. Durante la fase finale di rilascio, è stata condotta una rigorosa sessione di collaudo manuale End-to-End (E2E[G]), formalmente definita come *User Acceptance Testing* (UAT[G]).

Questa fase si è rivelata cruciale per valutare il comportamento dell'intero sistema Nexum in condizioni di utilizzo reale. Il collaudo ha previsto l'esecuzione di scenari operativi completi per certificare il soddisfacimento dei 5 Criteri di Accettazione (CA) concordati nel *Business Requirements Document*. I flussi testati hanno simulato il ciclo di vita integrale del dato: dal caricamento fisico di un Curriculum Vitae eterogeneo sull'interfaccia client, passando per l'attesa asincrona dell'elaborazione NLP su cloud (AWS Bedrock), fino alla verifica visiva della corretta instanziazione dei grafi relazionali (Radar e Timeline) e alla successiva conferma manuale delle competenze da parte di un utente con privilegi HR.

## 6.6 Risultati quantitativi e metriche di copertura

Al fine di verificare la reale efficacia della suite di test e validare la robustezza dell'applicazione secondo gli standard qualitativi aziendali, è stata eseguita un'analisi approfondita della copertura del codice (*code coverage*). I report estratti attestano un'esaustiva copertura logica sia dello strato backend (Ruby on Rails) sia dell'infrastruttura client frontend (Angular).

**Nota metodologica sul perimetro di test:** Si precisa che le metriche di copertura esposte in questa sezione si riferiscono esclusivamente alla base di codice implementata o modificata durante il tirocinio per il modulo di *Skill Intelligence*. I moduli *legacy* della piattaforma Nexum, già operativi, sono stati esclusi dal perimetro di test per isolare e valutare unicamente l'affidabilità del lavoro svolto. Nelle tabelle seguenti, i risultati di *coverage* sono espressi tramite soglie minime garantite anziché con percentuali esatte. Questa

scelta formale è motivata dall'impossibilità di simulare deterministicamente alcune casistiche limite legate alle dipendenze esterne (esplicitate nei paragrafi successivi); l'esposizione di un dato percentuale rigido sarebbe risultata fuorviante e meno indicativa rispetto a una soglia validata.

### 6.6.1 Metriche di copertura del backend

L'esecuzione della suite di test lato server ha interessato l'intera architettura logica e di trasporto HTTP dei moduli principali (gestione profili, skill, progetti ed esperienze). I dati quantitativi raccolti sono riepilogati nella Tabella 6.1.

Tabella 6.1: Percentuali di code coverage per l'infrastruttura di Backend (Ruby on Rails).

| **Componente Backend** | **Line** | **Function** | **Branch** |
|---|---|---|---|
| Controllers (Presentazione) | $> 85\%$ | $> 90\%$ | $> 80\%$ |
| Service Objects (Business Logic) | $> 90\%$ | $> 90\%$ | $> 90\%$ |

#### Criticità e scenari non coperti

È doveroso precisare che nell'ambito del progetto un *Branch Coverage* con un valore elevato sulla logica di business interna non si traduce nell'infallibilità assoluta del sistema. L'architettura sviluppata per Nexum dipende fortemente da servizi Cloud e modelli generativi esterni, i quali introducono fisiologicamente scenari limite (*edge cases*) non simulabili in modo deterministico all'interno di un ambiente di test isolato:

- **Imprevedibilità dell'LLM (Allucinazioni):** Sebbene il *System Prompt* forzi Claude Haiku a restituire esclusivamente payload JSON, i test unitari si basano su risposte *mockate* ideali. Non possono prevedere tutte le possibili "allucinazioni" del modello reale in presenza di CV formattati in modo atipico, che potrebbero variare imprevedibilmente la struttura dell'output.
- **Fallimenti strutturali nel parsing dei file:** I test coprono l'ingestione di file PDF e DOCX standard. Tuttavia, non è possibile simulare esaustivamente il comportamento del pre-processore (gemma *Docx*) di fronte a documenti gravemente corrotti o con codifiche binarie proprietarie anomale, i quali potrebbero causare eccezioni di sistema impreviste.
- **Concorrenza Transazionale (Race Conditions):** La suite collauda le transazioni isolate, ma non applica test di *stress* per la concorrenza estrema. Non è coperto, ad esempio, il caso limite in cui due responsabili HR tentino simultaneamente, nella medesima frazione di secondo, di assegnare la stessa *Skill* allo stesso dipendente, delegando la salvaguardia dell'integrità ai soli *unique constraints* del database PostgreSQL.

### 6.6.2 Metriche di copertura del frontend

I test del client Single Page Application, orchestrati tramite il framework *Jest*, sono stati divisi per isolare lo strato dei servizi di estensione V3 (incaricati dello scambio dati in formato JSON) dallo strato dei componenti visuali della vista. Le metriche aggregate sono riportate nella Tabella 6.2.

Tabella 6.2: Metriche di copertura estrattive per l'infrastruttura Frontend (Angular).

| **Componente Frontend** | **Statements** | **Branches** | **Functions** | **Lines** |
|---|---|---|---|---|
| Servizi (V3 Extensions) | $> 85\%$ | $> 80\%$ | $> 80\%$ | $> 85\%$ |
| Componenti della Vista | $> 85\%$ | $> 80\%$ | $> 85\%$ | $> 85\%$ |

#### Criticità e scenari non coperti

Nonostante l'elevata affidabilità raggiunta, il divario rimanente per il completamento totale delle metriche (in particolare per quanto riguarda la *Branch Coverage* attorno all'80-86%) è riconducibile ad alcuni limiti pratici e compromessi architetturali adottati in fase di collaudo:

- **Librerie di rendering e canvas HTML5:** Il collaudo profondo del DOM generato da librerie di *Data Visualization* (come ECharts all'interno del `SunburstChartComponent`) risulta instabile in un ambiente Node.js simulato (JSDOM). Per evitare crash causati dalla mancanza delle API Canvas native, il rendering di questi elementi visivi è stato inibito tramite sovrascrittura del template (*template override*), lasciando strutturalmente scoperte alcune *callback* interne di inizializzazione grafica.
- **Stati visivi transitori:** Lo sforzo di *testing* sui componenti si è concentrato sui flussi logici primari (Single Source of Truth). Di conseguenza, alcune funzioni minori legate a comportamenti puramente estetici (es. la gestione della chiusura manuale di *tooltip*, i *toggle* visivi di pannelli secondari o le transizioni di *loading* istantanee) non sono state asserite esplicitamente, abbassando marginalmente la copertura dei condizionali.
- **Scenari di rete estremi (*Edge Cases*):** Nello strato dei servizi, alcuni percorsi di intercettazione degli errori legati a errori critici del server o a risposte API strutturalmente malformate non sono stati simulati in modo esaustivo, privilegiando la copertura totale dei flussi nominali (*happy path*) e degli errori di validazione di business più frequenti.

# Capitolo 7

# Conclusioni

## 7.1 Raggiungimento degli obiettivi

In questa sezione si traccia un bilancio conclusivo del lavoro svolto, confrontando gli obiettivi prefissati in fase di analisi con i risultati effettivamente conseguiti.

### 7.1.1 Delimitazione del perimetro di intervento e fattori acceleranti

Per inquadrare correttamente la reale portata del lavoro individuale svolto nelle otto settimane di tirocinio formativo, è fondamentale delimitare il perimetro dell'intervento. La transizione da un semplice prototipo isolato a un ecosistema di livello aziendale completo e collaudato è stata resa possibile dall'innesto del nuovo modulo all'interno della piattaforma preesistente Nexum, già operativa, e dallo sfruttamento di precisi fattori tecnologici:

- **Ecosistema preesistente:** Il progetto si è innestato su una base strutturale solida. Aspetti infrastrutturali complessi, quali l'autenticazione degli utenti, la gestione del database multi-organizzazione, librerie di componenti per le interfacce frontend, nonché l'infrastruttura per il salvataggio di dati tramite bucket S3, risultavano già interamente configurati dal team ospitante. Ciò ha permesso di dedicare l'intero periodo di tirocinio esclusivamente allo sviluppo delle logiche verticali del modulo di *Skill Intelligence*.
- **Produttività del framework architetturale:** L'adozione di Ruby on Rails come tecnologia di *backend* ha costituito il principale fattore abilitante per la rapidità di esecuzione. Grazie al suo paradigma *Convention over Configuration* e all'impiego del pattern *Service Object*, è stato possibile orchestrare complesse logiche di modellazione dei dati ed esporre molteplici *endpoint* RESTful in tempi ridotti.
- **Transizione metodologica verso l'integrazione Cloud:** Il piano originario prevedeva lo sviluppo di elaborazioni in linguaggio Python per l'analisi del linguaggio naturale, un'attività che avrebbe richiesto tempistiche di addestramento incompatibili con la durata del tirocinio. La scelta architetturale di delegare l'estrazione semantica

alle API fornite da AWS Bedrock ha trasformato la criticità da una problematica di *Data Science* a una più agile attività di ingegnerizzazione dei *prompt* e di integrazione in ambiente Cloud, garantendo risultati nettamente superiori in tempistiche altamente ottimizzate.

- **Standardizzazione dell'interfaccia grafica:** Relativamente allo sviluppo lato *client*, l'impiego di librerie di componenti predefinite e del sistema di design interno dell'azienda ha ridotto drasticamente la necessità di produrre codice strutturale per la presentazione visiva. Questo approccio ha consentito di focalizzare l'attenzione ingegneristica unicamente sulla logica applicativa e sull'integrazione di strutture grafiche e interattive complesse.
- **Infrastruttura di collaudo preconfigurata:** Il conseguimento di elevate percentuali di copertura del codice è stato agevolato dalla presenza di un ambiente di collaudo già formalizzato e integrato nel progetto. Lo sforzo ingegneristico in tale ambito si è pertanto concentrato sulla stesura dei soli casi di test relativi alle funzionalità sviluppate durante il tirocinio, un'operazione resa particolarmente efficiente dal rigoroso isolamento a singola responsabilità garantito dai *Service Object*.

### 7.1.2 Consuntivo delle attività: Evoluzione rispetto al piano iniziale

Al netto dei fattori abilitanti appena descritti, nel corso dello svolgimento del tirocinio l'ambito dei lavori si è progressivamente ampliato e raffinato rispetto al documento di pianificazione originale. Di seguito vengono analizzati i principali scostamenti e gli arricchimenti funzionali apportati, formalizzati successivamente nel *Business Requirements Document*.

#### Da prototipo a ecosistema integrato

Il piano di lavoro originale prevedeva la realizzazione di un prototipo *stand-alone*, focalizzato prevalentemente sull'estrazione di skill dai CV tramite *parsing*. Durante la progettazione, questo perimetro è stato profondamente rivisto: le analisi hanno infatti portato alla definizione e implementazione di un'architettura completa divisa in cinque fasi incrementali, concepita per integrarsi in modo organico e sicuro, in conformità con le normative sulla protezione dei dati (GDPR), all'interno della piattaforma HR preesistente Nexum. In quest'ottica, l'utilizzo inizialmente previsto del linguaggio Python è stato scartato al fine di garantire un'integrazione nativa e maggiormente efficiente all'interno dell'ecosistema Ruby. Conseguentemente, i modelli di Intelligenza Artificiale non sono stati eseguiti in locale, ma invocati tramite chiamate API verso l'infrastruttura AWS, avvalendosi di librerie di interfacciamento integrate direttamente nel framework Ruby on Rails.

#### Interattività del profilo e Governance HR

Il piano originario contemplava la creazione di un generico "modello di profilo utente skill-based". A livello implementativo, poiché un'anagrafica utente era già presente nell'applicativo di base Nexum, l'intervento si è concentrato sull'estensione di tale entità. Il profilo preesistente è stato reso altamente dinamico, permettendo l'associazione di competenze strutturate attinte da un dizionario standardizzato. Al fine di garantire l'integrità dei dati, è stato inoltre implementato un fondamentale flusso di validazione manuale da parte del reparto HR, concepito come strategia concreta di mitigazione contro il rischio di informazioni incomplete o di "allucinazioni" semantiche introdotte dall'algoritmo cloud.

#### Motore di suggerimenti

L'implementazione di logiche per il suggerimento delle competenze era stata originariamente relegata tra gli obiettivi puramente "facoltativi". Durante lo sviluppo, l'importanza strategica di questa funzionalità è emersa chiaramente, portandola a diventare un pilastro centrale del modulo *Skill Matching Engine*. La disponibilità di profili utente standardizzati ha permesso di abilitare il calcolo algoritmico dell'aderenza tra un dipendente e un determinato ruolo. Qualora si rilevi un divario di competenze (*skill gap*), ma sia presente una percentuale di compatibilità sufficientemente elevata, il sistema interviene per colmare le lacune proponendo l'associazione con un altro dipendente per percorsi di tutoraggio aziendale (*mentoring*) oppure suggerendo attività formative autonome. Inoltre, per rispondere alle dinamiche di un ambiente aziendale in continua evoluzione, il motore supporta la mobilità interna valutando costantemente i progetti attivi per verificare se le attuali allocazioni delle risorse risultino ottimali.

#### Mappatura progetti e matching aziendale

L'espansione più significativa del progetto rispetto alle tempistiche preventivate riguarda l'introduzione del dominio della mappatura organizzativa. Mentre la pianificazione iniziale si arrestava a una basilare *gap analysis* sul singolo individuo, lo sviluppo effettivo ha portato alla realizzazione di una quinta fase architetturale inedita (Fase 5). Questa componente introduce il collegamento diretto tra le competenze disponibili e le reali esigenze di business. È stata infatti implementata un'interfaccia dedicata alla creazione e gestione di entità "Progetto", che consente di definire analiticamente i ruoli e le competenze necessarie al loro completamento. Sulla base di questi requisiti operativi, il sistema è in grado di elaborare graduatorie (*ranking*) per identificare i candidati maggiormente idonei, supportando in modo proattivo il trasferimento di risorse e l'assegnazione automatizzata del personale.

## 7.2 Maturazione professionale e conoscenze acquisite

L'esperienza di stage ha rappresentato un'opportunità fondamentale per la mia crescita professionale e formativa. Dal punto di vista prettamente tecnico, lo sviluppo del sistema mi

ha permesso di consolidare e ampliare in modo significativo le mie competenze su tecnologie e framework moderni, quali Ruby on Rails per l'infrastruttura backend e Angular per lo sviluppo del frontend. Inoltre, ho avuto l'occasione di approfondire sul campo ambiti fortemente innovativi come l'Intelligenza Artificiale e il Natural Language Processing (NLP), applicandoli concretamente per l'estrazione e la categorizzazione automatizzata delle informazioni dai CV.

Oltre all'ampliamento delle competenze tecniche, l'inserimento in un reale contesto aziendale ha segnato un importante traguardo personale rispetto all'ambiente puramente accademico. Ho imparato a gestire in modo efficace le responsabilità assegnatemi e a rispettare rigorose scadenze operative. La partecipazione attiva a meeting di allineamento e revisione mi ha inoltre permesso di affinare le dinamiche di interazione in un team di lavoro, comprendendo a fondo l'importanza della comunicazione per gestire e soddisfare al meglio le aspettative degli stakeholder.

## 7.3 Sviluppi futuri

Il sistema realizzato costituisce una base solida e scalabile che si presta a svariate prospettive di ampliamento. Disponendo di una mappatura digitalizzata e completa delle competenze e delle esperienze di ciascun dipendente, risulta naturale ipotizzare l'integrazione di strumenti avanzati per l'analisi delle potenzialità aziendali. In particolare, lo sviluppo di un modulo di *data analytics* orientato all'odierno mercato informatico permetterebbe di incrociare le competenze interne con i trend tecnologici emergenti, fornendo alla dirigenza insight preziosi per assumere strategiche e pianificare le direzioni di crescita futura.

Un ulteriore e promettente sviluppo consisterebbe nella realizzazione di un modulo dedicato alla ricerca attiva e intelligente di risorse esterne. Tale strumento, sfruttando la medesima intelligenza algoritmica implementata per l'analisi interna, potrebbe intercettare automaticamente nuovi talenti sul mercato e suggerire nuove opportunità di business per l'azienda o per colmare in modo mirato eventuali mancanze (*skill gap*) nell'organico attuale.

# Appendice A

# Pianificazione settimanale

## A.1 Tabella pianificazione settimanale

Tabella A.1: Pianificazione settimanale delle attività di stage basata sull'architettura a 5 fasi del BRD e relative metriche di avanzamento.

| **Settimana** | **Fase e Attività Pianificate** | **Rispetto Tempi** | **Note / Criticità** |
|---|---|---|---|
| **Settimana 1** | **Fase 1**: Setup ambiente full-stack (RoR, Angular, AWS).<br>Modellazione database PostgreSQL per anagrafica competenze e sviluppo backend HR. | Completato nei tempi | Setup architetturale complesso ma fondamentale per soddisfare i vincoli tecnologici. |
| **Settimana 2** | **Fase 1 e Inizio Fase 2**: Sviluppo dizionario competenze e gerarchie.<br>Implementazione modulo di upload CV nei formati supportati e studio NLP base. | Completato nei tempi | Necessaria molta attenzione nell'analisi della struttura dei diversi formati di curriculum. |
| **Settimana 3** | **Fase 2**: Parsing automatico del testo, estrazione skill e anni di esperienza.<br>Mapping sul dizionario standard, deduplicazione e ottimizzazione performance ($<$ 10 sec). | Lieve ritardo | Complessità elevata nell'algoritmo di normalizzazione e nel matching NLP, che ha richiesto test aggiuntivi. |
| **Settimana 4** | **Fase 3**: Sviluppo interfacce Angular per il Profilo Dinamico.<br>Workflow di validazione per l'HR. | Completato (Ritardo recuperato) | Ottima integrazione delle mitigazioni del rischio (validazione HR manuale per dati incompleti). |

*Continua nella pagina successiva...*

Tabella A.1: Pianificazione settimanale (continua)

| **Settimana** | **Fase e Attività Pianificate** | **Rispetto Tempi** | **Note / Criticità** |
|---|---|---|---|
| **Settimana 5** | **Fase 4**: Implementazione grafica dello Skill Radar.<br>Sviluppo logiche di analisi gap tra utente e ruolo e implementazione motore di suggerimenti (formazione, mobilità, mentoring). | Completato nei tempi | Fase ad alto valore aggiunto, implementazione fluida grazie alla solidità del database (Fase 1). |
| **Settimana 6** | **Fase 5**: Sviluppo modulo inserimento skill per progetti.<br>Creazione del motore di ranking e matching automatico tra risorsa e progetto aziendale. | Completato nei tempi | Sfida algoritmica nel pesare correttamente l'importanza delle singole skill durante il matching. |
| **Settimana 7** | **Fase 5 e Integrazione**: Dashboard per gap organizzativi aziendali.<br>Ottimizzazione per sincronizzazione *near real-time* e supporto multi-organizzazione. | Lieve anticipo | Le tecnologie AWS scelte in partenza hanno facilitato notevolmente la scalabilità orizzontale. |
| **Settimana 8** | **Collaudo finale**: Verifica dei 5 Criteri di Accettazione (CA 1-5) e test end-to-end.<br>Debug e stesura della documentazione tecnica e di tesi. | Completato nei tempi | Approvazione finale da parte degli stakeholder. Criteri di accettazione ampiamente superati. |

# Appendice B

# Analisi dei requisiti dettagliata

## B.1 Diagrammi dei casi d'uso

In questa sezione vengono riportati i diagrammi UML dei casi d'uso (*Use Case Diagrams*) relativi alle cinque macro-fasi evolutive del progetto. Questi schemi modellano visivamente le interazioni tra gli attori (primari e secondari) e le specifiche funzionalità erogate dal sistema Nexum.

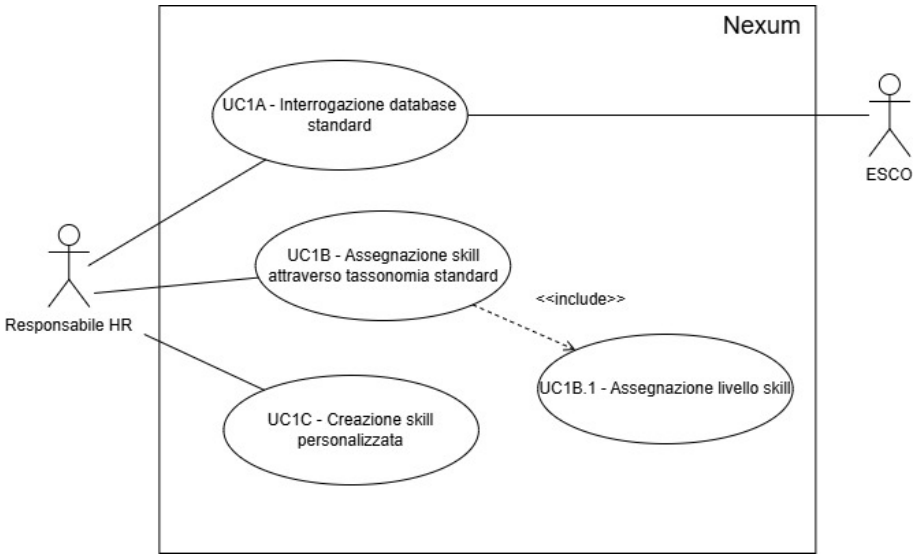


Figura B.1: Fase 1: Evoluzione Anagrafica Competenze.

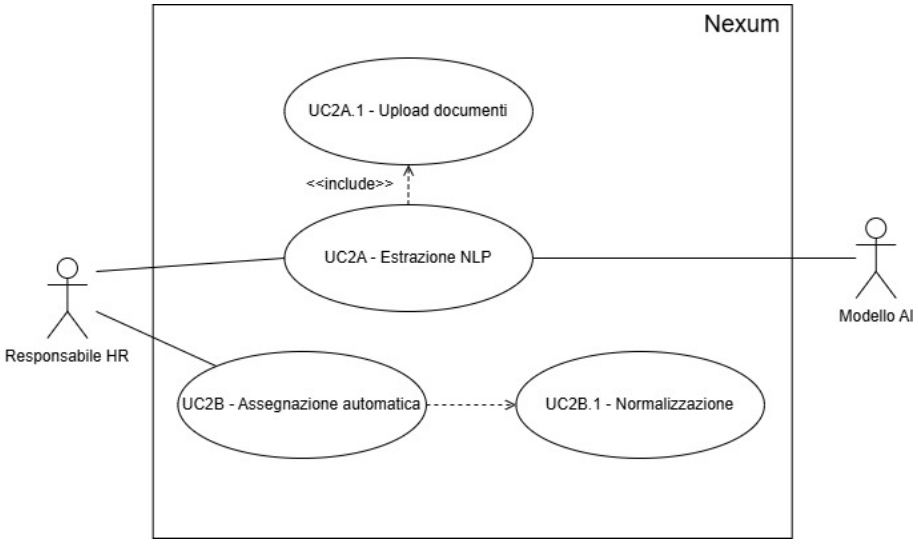


Figura B.2: Fase 2: Parsing CV e Data Extraction.

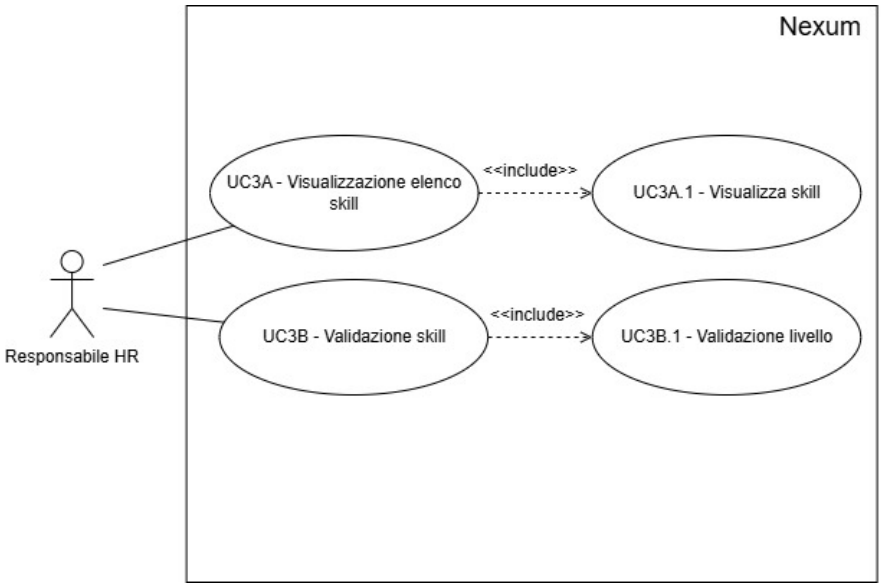


Figura B.3: Fase 3: Profilo Dinamico Risorsa.

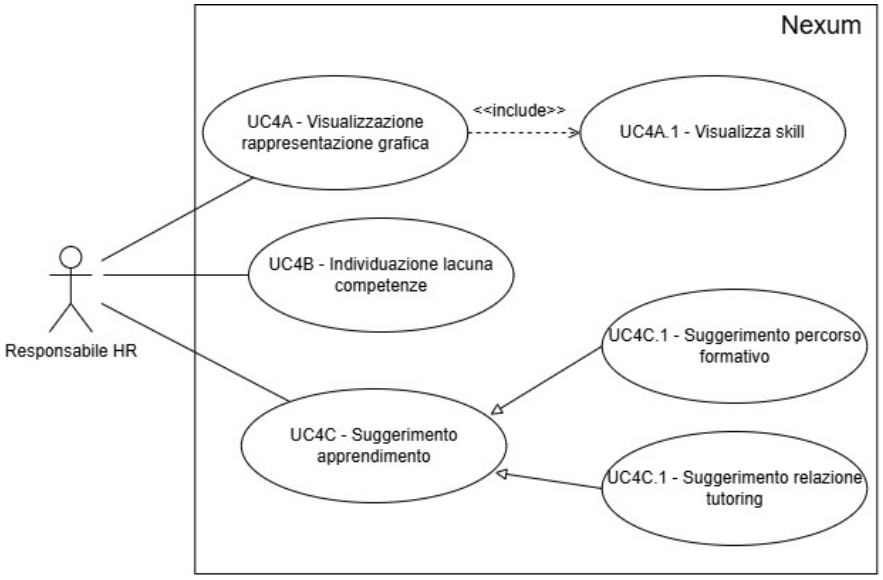


Figura B.4: Fase 4: Skill Radar Engine.

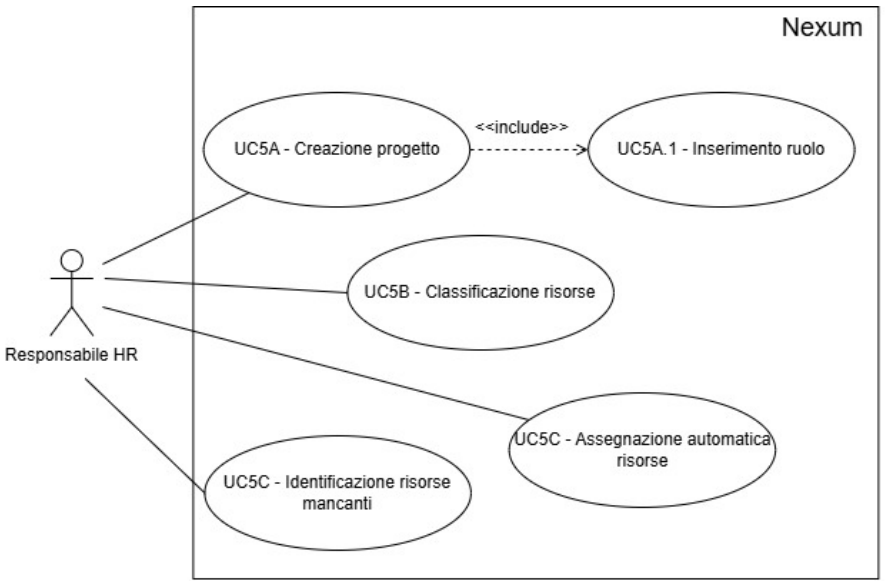


Figura B.5: Fase 5: Mappatura Progetti e Matching.

In questa sezione vengono riportati in modo esaustivo e tabellare tutti i requisiti identificati durante la fase di analisi per il modulo *Skill Radar* integrato nella piattaforma Nexum.

## B.2 Tabelle dei requisiti funzionali

### B.2.1 Spiegazione e legenda

Ogni requisito riportato nelle tabelle seguenti è identificato da un codice univoco nel formato **Rxyzz**, dove:

- **R** indica che si tratta di un Requisito.
- **x** indica che è un requisito funzionale o non funzionale.
- **y** rappresenta la priorità del requisito: **O** (Obbligatorio), **D** (Desiderabile), **F** (Facoltativo).
- **zz** è un numero progressivo a due cifre che parte da 00.

### B.2.2 Requisiti obbligatori

La seguente tabella elenca i requisiti funzionali di base estratti dagli ambiti funzionali del sistema.

Tabella B.1: Requisiti obbligatori di sistema

| **Codice** | **Descrizione** | **Stato** |
|---|---|---|
| RO00 | **Creazione skill**: Definizione nome, categoria e descrizione della competenza. | Rispettato |
| RO01 | **Associazione skill a utente**: Inserimento manuale da parte dell'HR. | Rispettato |
| RO02 | **Definizione livello competenza**: Configurazione di una scala di valutazione. | Rispettato |
| RO03 | **Categorizzazione skill**: Classificazione in transversal, cross-sector, sector-specific, occupation-specific. | Rispettato |
| RO04 | **Relazioni tra skill**: Definizione di correlazioni tra competenze. | Rispettato |
| RO05 | **Upload CV**: Caricamento documenti in formato PDF, DOC, DOCX. | Rispettato |
| RO06 | **Parsing testo**: Estrazione contenuto testuale dal CV. | Rispettato |
| RO07 | **Identificazione skill**: Riconoscimento automatico delle competenze. | Rispettato |
| RO08 | **Estrazione esperienza**: Individuazione anni e contesto di esperienza. | Rispettato |
| RO09 | **Mapping skill**: Allineamento a dizionario standard. | Rispettato |
| Continua nella pagina successiva... | | |

**Tabella B.1 – Continua dalla pagina precedente**

| Codice | Descrizione | Stato |
|---|---|---|
| RO10 | **Deduplicazione**: Rimozione duplicati. | Rispettato |
| RO11 | **Visualizzazione skill**: Elenco competenze e livelli. | Rispettato |
| RO12 | **Timeline esperienze**: Visualizzazione storico esperienze. | Rispettato |
| RO13 | **Validazione HR**: Revisione e conferma competenze. | Rispettato |
| RO14 | **Generazione radar**: Rappresentazione grafica delle competenze. | Rispettato |
| RO15 | **Confronto ruolo-skill**: Identificazione differenze tra competenze richieste e possedute. | Rispettato |
| RO16 | **Suggerimenti formazione**: Suggerimenti di percorsi formativi. | Rispettato |
| RO17 | **Mobilità interna**: Suggerimento di opportunità interne. | Rispettato |
| RO18 | **Mentoring**: Suggerimento di relazioni di apprendimento. | Rispettato |
| RO19 | **Inserimento skill richieste**: Definizione competenze per progetto o ruolo. | Rispettato |
| RO20 | **Ranking candidati**: Classifica delle risorse più adatte. | Rispettato |
| RO21 | **Matching automatico**: Associazione automatica risorsa-progetto. | Rispettato |
| RO22 | **Identificazione carenze**: Analisi delle competenze mancanti. | Rispettato |
| RO23 | **Interfaccia di validazione Human-in-the-loop**[G]: Modulo avanzato per HR finalizzato alla validazione manuale delle skill estratte, utile al riaddestramento continuo e alla correzione del modello. | Rispettato |

### B.2.3 Requisiti desiderabili

I requisiti desiderabili rappresentano funzionalità che apportano un notevole valore aggiunto al sistema, migliorandone le prestazioni o l'usabilità, ma la cui assenza non compromette il funzionamento di base delineato per il prototipo.

| Codice | Descrizione | Stato |
|---|---|---|
| RD00 | **Gestione progetti**: Creazione di un'interfaccia per la visualizzazione e la gestione dei progetti di un'azienda | Rispettato |
| RD01 | **Gestione ruoli**: Creazione di un'interfaccia per la visualizzazione e la gestione dei ruoli aziendali assumibili all'interno di un'azienda | Rispettato |

Tabella B.2: Requisiti desiderabili di sistema

### B.2.4 Requisiti facoltativi

I requisiti facoltativi includono funzionalità esplorative o di espansione futura del sistema, utili per definire la roadmap del prodotto ma non vincolanti per la fase attuale.

| Codice | Descrizione | Stato |
|---|---|---|
| RF00 | **Visualizzazione profilo utente**: Creazione di un'interfaccia per la visualizzazione delle informazioni dell'utente con grafici informativi | Rispettato |

Tabella B.3: Requisiti facoltativi di sistema

## B.3 Tabelle dei requisiti non funzionali

### B.3.1 Spiegazione e legenda

Ogni requisito riportato nelle tabelle seguenti è identificato da un codice univoco nel formato **Rxyzz**, dove:

- **R** indica che si tratta di un Requisito.
- **x** indica che è un requisito funzionale o non funzionale.
- **y** rappresenta la tipologia del requisito: **V** (Vincolo), **Q** (Qualità).
- **zz** è un numero progressivo a due cifre che parte da 00.

### B.3.2 Vincoli

I vincoli rappresentano le limitazioni tecnologiche, architetturali o organizzative entro le quali il sistema deve essere necessariamente progettato e sviluppato.

| Codice | Descrizione | Stato |
|---|---|---|
| RNV00 | **Integrazione architetturale**: Il modulo Skill Radar deve integrarsi in modo nativo all'interno dell'architettura preesistente della piattaforma aziendale Nexum. | Rispettato |
| RNV01 | **Stack Tecnologico**: Lo sviluppo deve rispettare le tecnologie standard adottate dall'azienda, utilizzando Ruby on Rails (RoR) per il backend, Angular per il frontend e PostgreSQL per la persistenza dei dati. | Rispettato |
| RNV02 | **Infrastruttura Cloud**: Il sistema e i servizi di intelligenza artificiale integrati (come i modelli LLM) devono appoggiarsi all'infrastruttura Amazon Web Services (AWS), nello specifico utilizzando AWS Bedrock per il motore NLP. | Rispettato |
| RNV03 | **Isolamento Multi-tenant**: L'architettura del database e le logiche di backend devono garantire il rigoroso isolamento dei dati per supportare in modo sicuro la gestione multi-organizzazione. | Rispettato |
| RNV04 | **Documentazione**: Deve essere fornita una documentazione dettagliata e adeguata trattante l'elaborato presentato. | Rispettato |

Tabella B.4: Vincoli di sistema

### B.3.3 Requisiti di qualità

I requisiti di qualità (o di prestazione) definiscono gli standard misurabili che il sistema deve rispettare per garantire un'esperienza utente fluida, affidabile e scalabile nel tempo.

| Codice | Descrizione | Stato |
|---|---|---|
| RNQ00 | **Performance di Parsing**: Il tempo di esecuzione per il parsing, l'estrazione e la normalizzazione delle skill da un singolo documento (CV) deve essere inferiore a dieci secondi. | Rispettato |
| RNQ01 | **Aggiornamento dati**: I calcoli del motore Skill Radar e l'aggiornamento dei profili e delle dashboard HR devono avvenire in modalità *near real-time* a seguito di una modifica al database. | Rispettato |
| RNQ02 | **Scalabilità elevata**: L'infrastruttura logica, in particolare la pipeline NLP, deve essere progettata per gestire picchi di carico scalando orizzontalmente (es. elaborazione parallela di upload massivi di documenti). | Rispettato |
| RNQ03 | **Usabilità e Adozione**: L'interfaccia utente (UI) deve risultare sufficientemente chiara e accessibile da favorire un'alta adozione da parte del team HR, riducendo la complessità di validazione manuale (mitigazione del rischio). | Rispettato |
| RNQ04 | **Manutentibilità e Documentazione**: La documentazione fornita deve essere facilmente comprensibile e orientata a uno sviluppo scalabile in caso di ampliamento del prodotto proposto. | Rispettato |

Tabella B.5: Requisiti di qualità e prestazione

# Appendice C

# Progettazione

## C.1 Schema UML classi: Nexum Skill Intelligence

Questa sezione illustra la progettazione del modulo preposto all'elaborazione automatizzata dei Curriculum Vitae e all'estrazione delle competenze. Il diagramma UML delle classi mappa il flusso di orchestrazione del sistema, evidenziando la netta separazione delle responsabilità (*Single Responsibility Principle*) ottenuta tramite l'impiego del pattern *Service Object*. Lo schema traccia il percorso logico dell'informazione: dalla ricezione della richiesta nel *Controller*, passando per l'esecuzione asincrona tramite *Job*, fino all'invocazione dei micro-servizi di interfacciamento con l'Intelligenza Artificiale (AWS Bedrock) e con le ontologie di dominio esterne (ESCO).

**Nota metodologica:** Al fine di garantire la massima leggibilità nel formato di stampa e di focalizzare l'attenzione sull'architettura logica ad alto livello, il diagramma riportato in Figura C.1 presenta una struttura semplificata. Per questo motivo, lo schema non rispetta fedelmente al 100% lo standard UML formale: sono stati volutamente omessi gli attributi interni delle singole classi, le firme dei metodi, le tipizzazioni dei dati e le interfacce di serializzazione. La modellazione si concentra volutamente sulle dipendenze dirette, sulle associazioni gerarchiche tra i moduli e sui punti di integrazione con i sistemi *third-party*. I colori utilizzati rappresentano i ruoli delle diverse classi all'interno dell'architettura: azzurro rappresenta il livello di presentazione, viola rappresenta il livello di business logic, giallo il livello di accesso ai dati e il bianco rappresenta l'utilizzo di un servizio esterno.

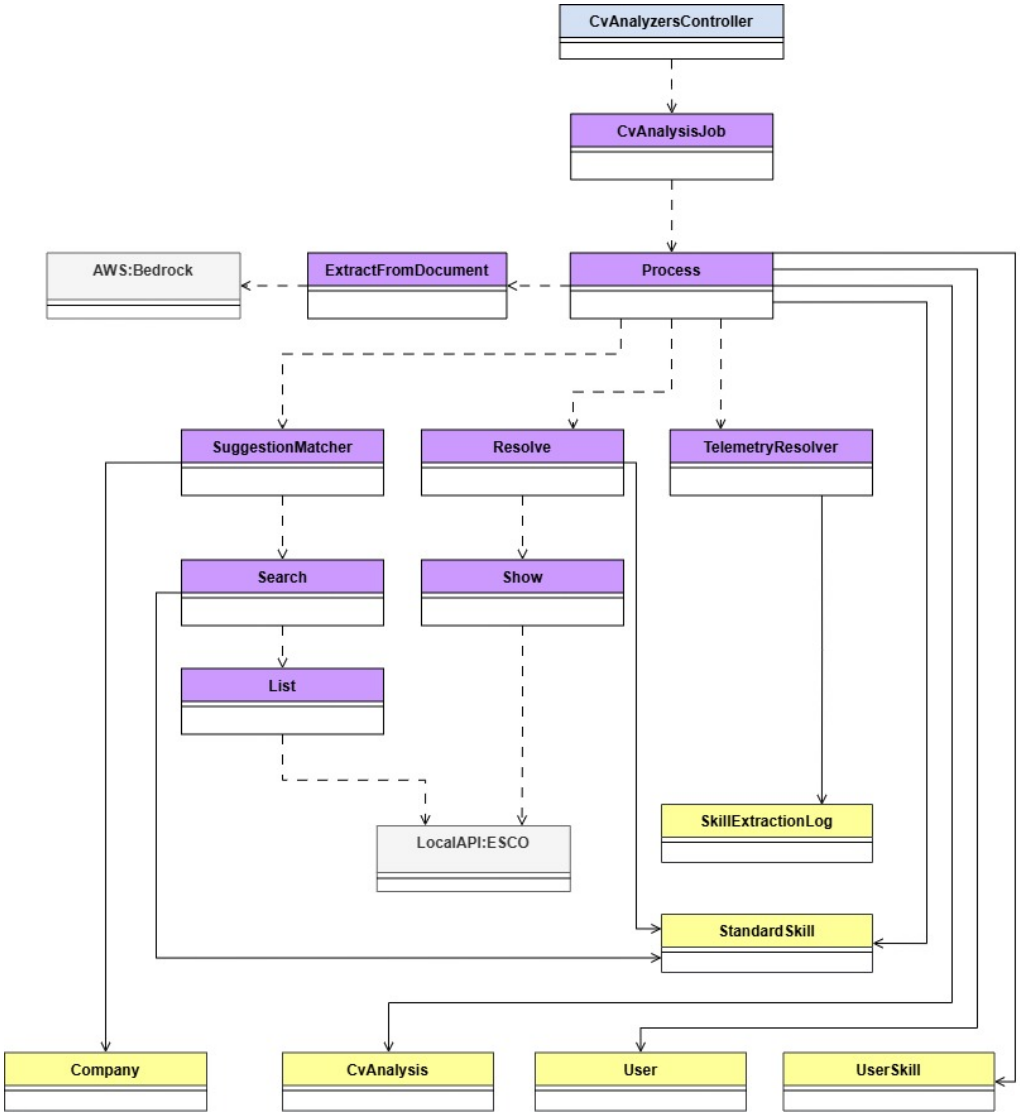


Figura C.1: Schema UML della pipeline di elaborazione CV e assegnazione skill

## C.2 Schema ER: Nexum Skill Radar

Il presente capitolo illustra la progettazione concettuale della base di dati, fondamento dell'architettura di persistenza del modulo *Nexum Skills Radar*. Lo schema Entità-Relazione (ER) proposto mappa visivamente le interazioni tra i domini logici principali del sistema: l'anagrafica degli utenti, il catalogo unificato delle competenze, la strutturazione dell'organigramma aziendale e i flussi di formazione e analisi documentale.

**Nota metodologica:** Al fine di preservare la chiarezza espositiva e garantire la massima leggibilità del diagramma all'interno del formato di stampa, lo schema riportato in Figura C.2 costituisce una rappresentazione macroscopica ad alto livello. Pertanto, esso non rispecchia fedelmente al 100% l'implementazione fisica finale nel database relazionale (PostgreSQL): sono stati volutamente omessi gli elenchi esaustivi degli attributi interni per

singola entità, le chiavi primarie/esterne di servizio, le marche temporali e alcune entità associative minori. La modellazione si concentra volutamente sull'evidenziare i legami logici, le dipendenze e le cardinalità tra i nodi del dominio applicativo.

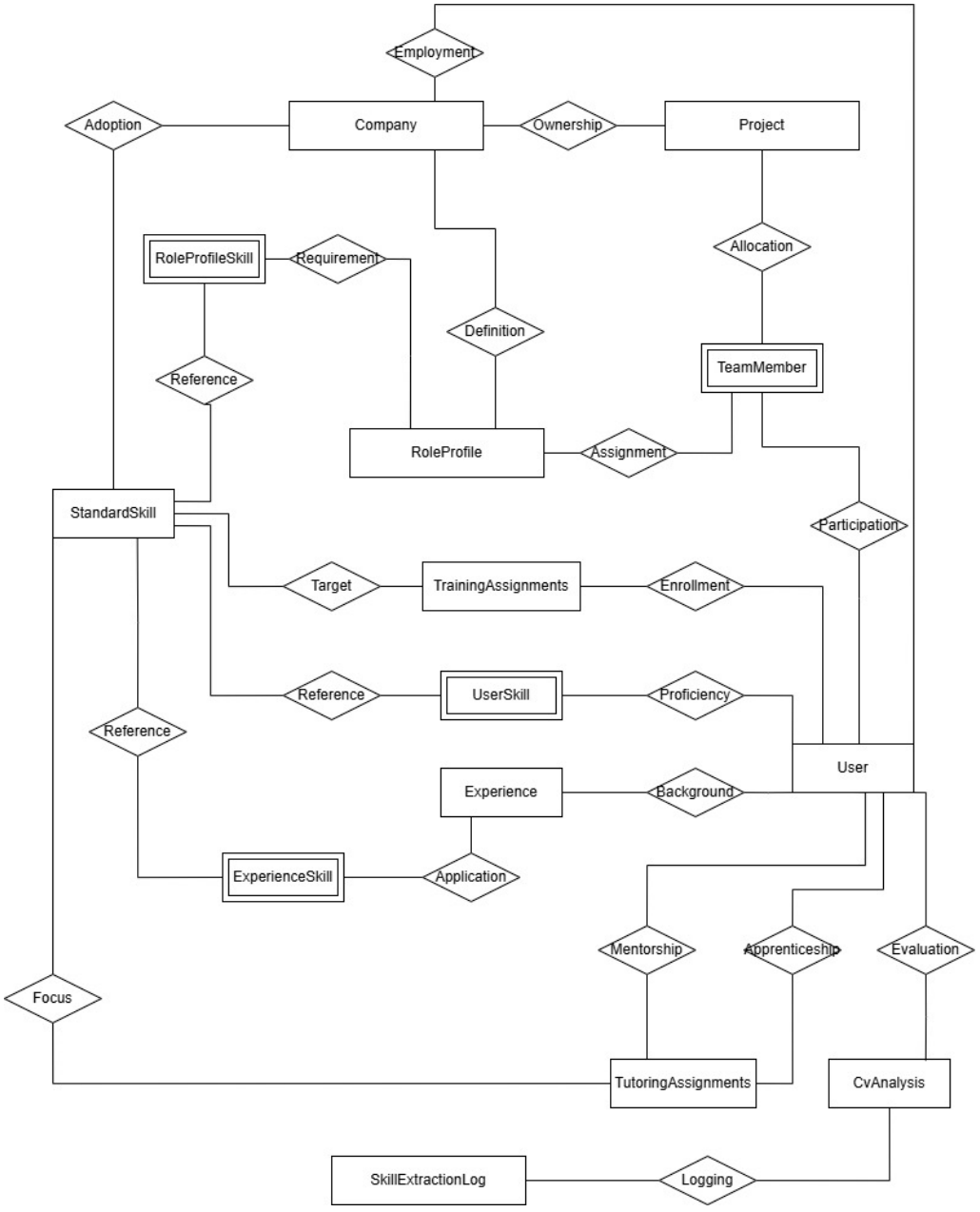


Figura C.2: Schema ER macroscopico: gestione competenze, esperienze e progetti

## C.3 Codice esempio: Nexum Skill Intelligence

Listing C.1: Servizio di orchestrazione AWS Bedrock per l'estrazione NLP

```
class ExtractFromDocument
  def initialize(file_content:, prompt:, mime_type: "application/pdf")
    @file_content = file_content
    @prompt = prompt
    @mime_type = mime_type
  end

  def call
    # 1. Inizializzazione client AWS Bedrock (credenziali omesse per
        brevita')
    client = Aws::BedrockRuntime::Client.new(...)
    # 2. Codifica del documento strutturata in base al MIME type
    content_block = if @mime_type == "application/pdf"
                      {
                        type: "document",
                        source: {
                          type: "base64",
                          media_type: @mime_type,
                          data: Base64.strict_encode64(@file_content)
                        }
                      }
                    else
                      { type: "text", text: "<cv>\n#{@file_content}\n</cv>"
                         }
                    end
    # 3. Composizione del payload con iniezione del System Prompt
    payload = {
      anthropic_version: "bedrock-2023-05-31",
      max_tokens: 3000,
      system: @prompt.strip,
      messages: [
        {
          role: "user",
          content: [
            content_block,
            { type: "text", text: "Estrai i dati richiesti in formato JSON.
                " }
          ]
        }
      ]
    }
    # 4. Invocazione del modello LLM (Claude Haiku 4.5)
    response = client.invoke_model({
      model_id: "eu.anthropic.claude-haiku-4-5-20251001-v1:0",
      content_type: "application/json",
      accept: "application/json",
      body: payload.to_json
    })
    # 5. Sanitizzazione e isolamento del JSON puro tramite Regex
    response_body = JSON.parse(response.body.read)
    extracted_text = response_body.dig("content", 0, "text")
    json_match = extracted_text.match(/\{.*\}/m)
    clean_json = json_match ? json_match[0] : extracted_text
    { success: true, data: JSON.parse(clean_json) }
  rescue StandardError => e
    # Gestione delle eccezioni (omessa per brevita')
    { success: false, errors: [e.message] }
  end
end
```

# Bibliografia e Sitografia


[1] Beck, K. et al., *Manifesto for Agile Software Development*, 2001. URL: http://agilemanifesto.org/

[2] Academy to Innovate HR (AIHR), *Organizational Network Analysis (ONA): A Complete Guide*. URL: https://www.aihr.com/blog/organizational-network-analysis/

[3] Amazon Web Services (AWS), *Amazon S3 Cloud Storage*. URL: https://aws.amazon.com/pm/serv-s3/

[4] Amazon Web Services (AWS), *Amazon S3 User Guide*. URL: https://docs.aws.amazon.com/AmazonS3/latest/userguide/Welcome.html

[5] Amazon Web Services (AWS), *AWS Whitepapers & Guides*. URL: https://aws.amazon.com/whitepapers/

[6] Amazon Web Services (AWS), *What is Amazon Bedrock?*. URL: https://docs.aws.amazon.com/bedrock/latest/userguide/what-is-bedrock.html

[7] Angular, *Angular Official Documentation - Overview*. URL: https://angular.dev/overview

[8] Anthropic, *Claude 4.5 Haiku*. URL: https://www.anthropic.com/news/claude-haiku-4-5

[9] Anthropic, *Build with Claude: PDF Support*. URL: https://platform.claude.com/docs/en/build-with-claude/pdf-support

[10] Apache Software Foundation, *Apache ECharts Official Documentation*. URL: https://echarts.apache.org/en/index.html

[11] Devlin, J. et al., *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*, arXiv:1810.06818, 2018. URL: https://arxiv.org/html/1810.06818v3

[12] Atlassian Agile Coach, *Types of Software Testing*. URL: https://www.atlassian.com/continuous-delivery/software-testing/types-of-software-testing

[13] Bruno API Client, *Official Website*. URL: https://www.usebruno.com/

[14] European Commission, *What is ESCO?*. URL: http://esco.ec.europa.eu/en/about-esco/what-esco

[15] European Commission, *ESCO Linked Open Data*. URL: http://esco.ec.europa.eu/en/use-esco/linked-open-data

[16] Gartner, *Human Resources Glossary*. URL: https://www.gartner.com/en/human-resources/glossary

[17] GeeksforGeeks, *Named Entity Recognition (NER)*. URL: https://www.geeksforgeeks.org/nlp/named-entity-recognition/

[18] Git, *About Git*. URL: http://git-scm.com/about

[19] Chacon, S., Straub, B., *Pro Git Book* (Italian Version). URL: http://git-scm.com/book/it/v2

[20] IBM, *Cos'è l'elaborazione del linguaggio naturale (NLP)?*. URL: https://www.ibm.com/it-it/think/topics/natural-language-processing

[21] IBM, *Cloud Education - Database and AI Topics*. URL: https://www.ibm.com/topics

[22] Mozilla Developer Network (MDN), *Working with JSON*. URL: https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Objects/JSON

[23] Mozilla Developer Network (MDN), *REST Glossary*. URL: https://developer.mozilla.org/en-US/docs/Glossary/REST

[24] Mozilla Developer Network (MDN), *Developer Glossary*. URL: https://developer.mozilla.org/en-US/docs/Glossary

[25] Object Management Group (OMG), *UML Standards*. URL: https://www.omg.org/spec/UML/

[26] PostgreSQL Global Development Group, *About PostgreSQL*. URL: https://www.postgresql.org/about/

[27] PostgreSQL Global Development Group, *PostgreSQL Official Website*. URL: https://www.postgresql.org/

[28] ReactiveX, *ReactiveX Documentation*. URL: https://reactivex.io/

[29] Narendra G O e Hashwanth S., *Named Entity Recognition based Resume Parser and Summarizer*, ResearchGate. URL: https://www.researchgate.net/publication/359694475_Named_Entity_Recognition_based_Resume_Parser_and_Summarizer

[30] Ruby Programming Language, *Official Website*. URL: https://www.ruby-lang.org/en/

[31] Ruby on Rails Guides, *Using Rails for API-only Applications*. URL: https://guides.rubyonrails.org/api_app.html

[32] Stasko, J., Catrambone, R., *An evaluation of space-filling information visualizations for depicting hierarchical structures*, Semantic Scholar. URL: https://www.semanticscholar.org/paper/An-evaluation-of-space-filling-information-for-Stasko-Catrambone/c4123c56a7c593a2e0b0fb051117e6df1e9bdad6

[33] Microsoft / TypeScript, *TypeScript Official Documentation*. URL: https://www.typescriptlang.org/docs/

[34] Ultralytics, *Named Entity Recognition (NER) Glossary*. URL: https://www.ultralytics.com/it/glossary/named-entity-recognition-ner

[35] Microsoft, *Visual Studio Code Official Website*. URL: https://code.visualstudio.com/

# Glossario

Il presente glossario è suddiviso in aree semantiche per agevolare la consultazione dei termini tecnici appartenenti ai diversi domini trattati nell'elaborato.

Nota terminologica: all'interno del sistema, il dominio di business delle Risorse Umane (HR) definisce i collaboratori come 'Risorse' o 'Dipendenti', entità che a livello architetturale e di database sono mappate in modo univoco dal modello 'User'

## Architettura software e tecnologie

**ACID:** (Atomicity, Consistency, Isolation, Durability). Insieme di proprietà architetturali che garantiscono l'affidabilità e la sicurezza delle transazioni all'interno di database relazionali.

**Active Record:** Pattern architetturale e modulo fondamentale di Ruby on Rails che implementa l'ORM (Object-Relational Mapping), semplificando la modellazione e l'interazione con il database relazionale PostgreSQL.

**Amazon S3 (o Object Storage):** Servizio di archiviazione in cloud offerto da AWS che gestisce i dati come "oggetti" all'interno di "bucket" (contenitori), utilizzato nel progetto per la memorizzazione permanente e sicura dei Curriculum Vitae.

**Angular:** Framework open-source sviluppato da Google, basato su TypeScript, utilizzato per la creazione di *Single Page Application* reattive e architetture client-side complesse.

**API:** (Application Programming Interface). Interfaccia software che consente a diverse applicazioni di comunicare tra loro. Nel contesto del progetto, le API REST sono utilizzate per far comunicare il client Angular con il server Ruby on Rails e per la connessione al servizio cloud AWS Bedrock.

**AR/VR:** (Augmented Reality / Virtual Reality). Tecnologie immersive per l'interazione digitale (Realtà Aumentata e Realtà Virtuale), settori in cui opera l'azienda per la progettazione di soluzioni software digitali.

**AWS:** (Amazon Web Services). Suite di servizi offerta da Amazon.

**Background Job (o Worker):** Processo eseguito in modo asincrono rispetto al thread principale dell'applicazione (es. per l'elaborazione NLP dei CV), necessario per non bloccare l'interfaccia utente durante operazioni computazionalmente pesanti.

**CRUD:** (Create, Read, Update, Delete). Acronimo che raggruppa le quattro operazioni fondamentali per la gestione persistente dei dati all'interno di un database o esposte tramite API RESTful.

**DOM:** (Document Object Model). Interfaccia di programmazione per documenti web (HTML) che ne rappresenta la struttura logica ad albero e permette agli script di manipolarne dinamicamente contenuto e stile.

**E2E:** (End-to-End). Tipologia di collaudo software che verifica il corretto funzionamento dell'intero flusso applicativo dall'inizio alla fine, simulando scenari d'uso reali per l'utente, dall'interfaccia client fino alla persistenza in database.

**GUI:** (Graphical User Interface). Interfaccia utente grafica che consente l'interazione con il sistema software attraverso elementi visivi (finestre, bottoni, icone) piuttosto che tramite riga di comando.

**HTML:** (HyperText Markup Language). Linguaggio di marcatura standard utilizzato per la definizione, la creazione e la strutturazione semantica delle pagine web.

**HTTP:** (Hypertext Transfer Protocol). Protocollo a livello di applicazione utilizzato per la trasmissione di documenti e dati ipertestuali sul World Wide Web.

**IoT:** (Internet of Things). Estensione della connettività di rete a dispositivi e oggetti fisici, permettendo loro di raccogliere, elaborare e scambiare dati con altri sistemi.

**JSON:** (JavaScript Object Notation). Formato standard aperto, leggero e basato su testo, utilizzato per la serializzazione e lo scambio di dati strutturati tra il server backend e l'interfaccia frontend.

**Mocking:** Tecnica di testing software che consiste nel sostituire oggetti o servizi di rete reali con componenti simulati (*mock objects*), al fine di isolare l'ambiente di test e validare la logica senza latenze o dipendenze esterne.

**Multi-tenant:** Architettura software in cui una singola istanza dell'applicazione viene eseguita su un server per servire contemporaneamente molteplici organizzazioni clienti (*tenant*), garantendo al contempo il rigoroso isolamento logico dei dati.

**MVVM:** (Model-View-ViewModel). Pattern architetturale che separa nettamente lo sviluppo dell'interfaccia utente (View) dalla logica di business (Model), impiegando un modello della vista (ViewModel) incaricato della gestione reattiva dello stato.

**ORDBMS:** (Object-Relational Database Management System). Sistema di gestione di database relazionale che integra funzionalità tipiche della programmazione orientata

agli oggetti (come l'ereditarietà e i tipi di dato complessi), di cui PostgreSQL è l'esempio principale.

**ORM:** (Object-Relational Mapping). Tecnica di programmazione (implementata tramite il modulo *Active Record*) che consente di interrogare e manipolare i database relazionali utilizzando il paradigma della programmazione orientata agli oggetti.

**PostgreSQL:** Sistema avanzato di gestione di database relazionale ad oggetti (ORDBMS) open-source, scelto per la persistenza transazionale e l'integrità dei dati del backend.

**REST:** (Representational State Transfer). Stile architetturale per sistemi distribuiti. Un'interfaccia RESTful espone le proprie risorse tramite metodi HTTP standard, fungendo da paradigma primario per le API del modulo backend.

**Ruby on Rails:** (Spesso abbreviato in Rails o RoR). Framework di sviluppo web full-stack scritto in linguaggio Ruby, utilizzato nel progetto in modalità API-only per la gestione del backend.

**RxJS:** (Reactive Extensions for JavaScript). Libreria per la programmazione reattiva che utilizza gli *Observable*, impiegata in Angular per la gestione asincrona degli eventi e delle chiamate HTTP.

**SPA:** (Single Page Application). Applicazione web che interagisce con l'utente aggiornando dinamicamente la singola pagina corrente e scambiando dati in *background*, garantendo un'esperienza utente fluida.

**SQL:** (Structured Query Language). Linguaggio standardizzato per l'interrogazione, la gestione e la manipolazione delle basi di dati relazionali.

**Stubbing (o Stub):** Tecnica di testing strettamente legata al Mocking; consiste nel sostituire una chiamata di rete o un metodo con una risposta predefinita e simulata, utilizzata per testare l'infrastruttura senza interrogare le API reali.

**TypeScript:** Superset sintattico di JavaScript sviluppato da Microsoft che aggiunge la tipizzazione statica opzionale e costrutti orientati agli oggetti avanzati.

**UAT:** (User Acceptance Testing). Ultima fase del ciclo di collaudo in cui il cliente o l'utente finale verifica il sistema software tramite scenari operativi completi per certificarne l'aderenza ai Criteri di Accettazione.

**UI:** (User Interface). Interfaccia utente, ovvero l'insieme dei componenti grafici o testuali attraverso cui un operatore umano interagisce con un software.

**UML:** (Unified Modeling Language). Linguaggio di modellazione standardizzato utilizzato nell'ingegneria del software per specificare, visualizzare e documentare gli artefatti del sistema (diagrammi delle classi, casi d'uso, flussi).

**URI / URL:** (Uniform Resource Identifier / Uniform Resource Locator). Stringhe di caratteri utilizzate rispettivamente per identificare e localizzare in modo univoco una risorsa (es. un endpoint o un documento) su una rete.

## Intelligenza artificiale e analisi dati

**AI:** (Artificial Intelligence). Disciplina dell'informatica che si concentra sullo sviluppo di algoritmi e modelli capaci di emulare abilità cognitive complesse tipiche dell'essere umano (apprendimento, ragionamento e comprensione del linguaggio).

**AWS Bedrock:** Servizio cloud gestito fornito da Amazon Web Services, che permette l'orchestrazione sicura e l'accesso tramite API a modelli fondativi di intelligenza artificiale.

**ESCO:** (European Skills, Competences, Qualifications and Occupations). Classificazione multilingue europea delle abilità, competenze e professioni, utilizzata come dizionario tassonomico standard per la normalizzazione semantica.

**LLM:** (Large Language Model). Modello di linguaggio generativo basato su reti neurali, addestrato su enormi volumi di dati, capace di inferire contesti e generare output strutturati in linguaggio naturale.

**NER:** (Named Entity Recognition). Sotto-disciplina dell'elaborazione del linguaggio naturale mirata a individuare, isolare e classificare entità specifiche e concetti chiave all'interno di documenti non strutturati.

**NLP:** (Natural Language Processing). Ramo dell'intelligenza artificiale che studia l'interazione tra computer e linguaggio umano, applicato nel progetto per automatizzare il *parsing* dei Curriculum Vitae.

**Prompt Engineering:** Pratica di progettazione e ottimizzazione degli input (prompt) forniti a un *Large Language Model* al fine di ottenere output precisi, strutturati e deterministici, essenziale per limitare le "allucinazioni" dell'IA.

**Skills Intelligence:** Quadro strategico e tecnologico volto a mappare, quantificare e analizzare le competenze del capitale umano in modo dinamico e guidato dai dati, supportando i processi aziendali di *Workforce Planning* e *Skill Gap Analysis*.

**System Prompt:** Istruzione di base fornita a un modello di intelligenza artificiale che ne definisce il ruolo, il perimetro d'azione e le regole logiche (come i vincoli di formattazione JSON) prima di elaborare le richieste dell'utente.

## Dominio applicativo (HR) e visualizzazione

**BRD:** (Business Requirements Document). Documento formale stilato all'inizio del progetto che delinea i requisiti funzionali, le fasi di sviluppo e gli obiettivi di business che il software deve soddisfare.

**CA:** (Criteri di Accettazione). Insieme di condizioni oggettive e scenari predefiniti che il software sviluppato deve superare positivamente per essere considerato concluso e pronto per la messa in produzione.

**CEO:** (Chief Executive Officer). Amministratore delegato, la più alta figura dirigenziale all'interno di un'azienda, responsabile della pianificazione e dell'approvazione delle macro-strategie operative o dei nuovi progetti.

**CV:** Curriculum Vitae. Documento informale e non strutturato, caricato dall'utente, che funge da sorgente dati grezza per la pipeline di estrazione.

**ECharts:** (Apache ECharts). Potente libreria open-source basata su JavaScript per la visualizzazione dei dati, utilizzata nel progetto per il rendering dei grafi interattivi.

**GDPR:** (General Data Protection Regulation). Regolamento europeo sul trattamento dei dati personali; nel contesto HR impone rigidi vincoli di sicurezza, isolamento (multi-tenant) e minimizzazione della persistenza di dati sensibili.

**Hard Skill / Soft Skill:** Rispettivamente, le competenze tecniche/metodologiche quantificabili e le competenze relazionali/trasversali (es. leadership, problem solving), entrambe cruciali per la profilazione completa della risorsa.

**Human-in-the-loop:** Approccio in cui l'intervento umano (in questo caso, la validazione da parte del personale HR) viene integrato nei processi decisionali automatizzati dall'intelligenza artificiale, garantendo la correttezza formale del dato estratto.

**ONA:** (Organizational Network Analysis). Metodologia analitica utilizzata in ambito HR per mappare e visualizzare le relazioni formali e informali (dinamiche sociali e di collaborazione) all'interno di un'organizzazione.

**Role Profile:** Profilo di Ruolo. Insieme strutturato di aspettative, responsabilità e competenze richieste (fabbisogno) affinché un dipendente possa ricoprire una specifica mansione aziendale.

**Skill Gap:** Divario misurabile tra le competenze attualmente possedute da una risorsa e quelle idealmente richieste dal suo ruolo o da un progetto specifico.

**Skill Matching Engine:** Componente logico-algoritmico del backend incaricato di calcolare in tempo reale i gap di competenza, clusterizzare l'idoneità delle risorse e generare ranking automatizzati per l'allocazione ottimale dei talenti sui progetti.

**Skill Radar:** Rappresentazione visiva a forma poligonale (*Radar Chart*) delle competenze, che permette un confronto geometrico immediato dei livelli di padronanza.

**Sunburst Chart:** Grafico radiale multi-livello utilizzato per rappresentare dati gerarchici, impiegato per mostrare la tassonomia delle competenze (dalle macro-categorie alle skill specifiche).

**Timeline:** Componente grafico di tipo cronologico che mappa linearmente eventi sequenziali, utilizzato per storicizzare l'evoluzione formativa e professionale del dipendente.

**Workforce Planning:** Processo strategico di pianificazione delle risorse umane, mirato ad allineare il capitale umano aziendale (assunzioni, mobilità interna, formazione) con gli obiettivi di business.