Università degli Studi di Padova

Dipartimento di Matematica "Tullio Levi-Civita"

Corso di Laurea in Informatica

SmartProspect: ecosistema distribuito per la gestione di dati aziendali

Tesi di Laurea

Relatore
Prof. Tullio Vardanega

Laureando
Diana Georgescu
Matricola 2075549

Anno Accademico 2025-2026

Sommario

L'elaborato analizza la progettazione e lo sviluppo di SmartProspect, una piattaforma *software* per la raccolta, l'integrazione e la gestione centralizzata di dati aziendali provenienti da fonti eterogenee. Approfondisce i processi di *crawling*, *scraping* e normalizzazione dei dati, l'utilizzo di architetture basate su *container* e code asincrone, nonché la realizzazione di interfacce *web* dedicate alla consultazione, alla reportistica e all'analisi delle informazioni raccolte. Descrive inoltre l'integrazione di strumenti di automazione, *Open-Source Intelligence*$_G$ e intelligenza artificiale all'interno di un ecosistema distribuito orientato alla *business intelligence*$_G$.

Struttura del documento. Il documento si articola in quattro capitoli. Il Capitolo 1 presenta il contesto aziendale di Spazio Dev, lo *stack* tecnologico adottato e i processi interni di sviluppo. Il Capitolo 2 illustra le motivazioni del tirocinio, gli obiettivi concordati e la loro pianificazione temporale. Il Capitolo 3 descrive il progetto SmartProspect, dall'architettura complessiva alle singole funzionalità implementate. Il Capitolo 4 offre una valutazione retrospettiva del percorso svolto, in relazione agli obiettivi prefissati e alle competenze acquisite.

Convenzioni tipografiche. Il *corsivo* segnala ogni termine usato in lingua diversa dall'italiano. Per le immagini e i dati tratti da terzi, la fonte è indicata nella didascalia corrispondente; per i materiali prodotti autonomamente nel corso dello *stage*, la didascalia riporta la dicitura `elaborazione propria`.

Indice

Elenco delle figure

Elenco delle tabelle

Elenco dei codici sorgenti

Glossario

Agile Metodologia di sviluppo *software* caratterizzata da un approccio iterativo e incrementale, che favorisce la collaborazione tra *team* e la risposta ai cambiamenti. 23

API *Application Programming Interface*, indica un insieme di definizioni e protocolli per la creazione e l'integrazione di applicazioni *software*. 5

business intelligence Comprende tipicamente processi di raccolta, integrazione, analisi e presentazione di dati aziendali, con l'obiettivo di identificare opportunità di mercato, ottimizzare processi e aumentare la competitività. iii

MVC *Model-View-Controller*, è un *pattern* architetturale che separa la logica applicativa, la presentazione e la gestione dei dati in tre componenti distinte. 5

OSINT Intelligenza di origine aperta, si riferisce all'insieme di tecniche e strumenti utilizzati per raccogliere informazioni disponibili pubblicamente con l'obiettivo di analizzarle e derivarne conclusioni utili. iii

RAG Tecnica di intelligenza artificiale che combina il recupero di informazioni da fonti esterne con la generazione di contenuti testuali. 22

TALL stack È una combinazione di tecnologie per lo sviluppo *web full-stack*, progettata specificamente per l'ecosistema Laravel. Consente di creare applicazioni moderne, reattive e dinamiche scrivendo pochissimo codice JavaScript e rimanendo quasi interamente all'interno di PHP. Comprende Tailwind CSS, Alpine.js, Laravel e Livewire. 6

UML Linguaggio visivo standardizzato utilizzato per progettare, documentare e descrivere l'architettura e il comportamento dei sistemi complessi. 24

Capitolo 1

Spazio Dev: il contesto lavorativo

Questo Capitolo presenta il contesto organizzativo e produttivo in cui ho svolto lo *stage*: il profilo dell'azienda ospitante, la tipologia di clientela servita, le tecnologie su cui si fonda il suo *stack* applicativo, i processi interni di sviluppo e, in chiusura, la sua propensione verso l'innovazione.

1.1 Introduzione all'azienda

Le informazioni che ho riportato in questo Capitolo derivano da due fonti complementari: la consultazione del sito *web* dell'azienda e alcuni colloqui informali con i fondatori e i colleghi, svolti nelle prime settimane di *stage* per comprendere il contesto produttivo in cui mi sarei inserita. Si tratta quindi di informazioni di prima mano, ma non verificate tramite documentazione ufficiale terza; i dati riportati vanno pertanto considerati accurati nella sostanza ma non necessariamente esaustivi.

Spazio Dev è una società di servizi IT e consulenza informatica con sede a Tombolo (PD), fondata nel 2023. Tra le attività osservate durante lo *stage* rientrano lo sviluppo di applicazioni *web*, la realizzazione di sistemi gestionali, l'integrazione di soluzioni basate su intelligenza artificiale e servizi di automazione dei processi.

La clientela di Spazio Dev comprende sia piccole attività locali sia aziende di dimensioni più consistenti, operanti in settori eterogenei: dalla ristorazione all'organizzazione di eventi, fino a realtà industriali di maggiori dimensioni.

Il comune denominatore delle richieste osservate è la necessità di migliorare l'efficienza dei processi interni e l'efficacia degli investimenti in *marketing* tramite strumenti digitali su misura.

Gli ambiti applicativi osservati durante il tirocinio sono trasversali e comprendono sia sviluppo *software* che interventi di natura più *hardware-oriented*:

realizzazione di sistemi gestionali e siti *web*, campagne di comunicazione automatizzata (email, WhatsApp), *bot* e automazioni, ma anche programmazione a basso livello di dispositivi (schermi digitali, *badge*, *firmware* per microcontrollori ESP, controllo di stampanti 3D). Tra le piattaforme sviluppate rientra SmartProspect, il progetto su cui si è concentrato il lavoro di tirocinio, descritto nel Capitolo 3.

1.2 *Stack* tecnologico aziendale

Di seguito illustro le principali tecnologie che costituiscono lo *stack standard* dell'azienda, con attenzione non solo al loro ruolo funzionale ma anche alle ragioni che ne hanno orientato l'adozione rispetto ad alternative disponibili.

Tecnologia	**Ruolo**	**Motivazione principale**
***Backend* e logica applicativa**		
PHP / Laravel	*Backend* applicativo	Continuità con lo *stack* preesistente; *framework* completo (*routing*, autenticazione, ORM) senza dipendenze esterne.
Presentazione e interfaccia utente (TALL *stack*)		
Blade	*Template engine*	Motore nativo di Laravel; ereditarietà dei *layout* e riuso dei componenti via *slot*.
Livewire	Componenti reattivi lato *server*	Interfacce dinamiche senza *endpoint* API REST dedicati; *framework* di base su cui è costruito Filament.
Filament	Interfacce amministrative	Licenza *open source* ed ecosistema di *plugin* in crescita.
Tailwind CSS / Alpine.js	Stilizzazione e interattività	Integrazione nativa in Filament; opzione leggera per componenti semplici.
Elaborazione asincrona e persistenza dati		
Laravel Horizon	Gestione code asincrone	*Dashboard* di monitoraggio e priorità configurabili sui *job*.
MariaDB	Persistenza dati	Compatibilità nativa con Eloquent; integrità referenziale per dati interconnessi.
Infrastruttura e *deployment*		
Docker	Containerizzazione	Isolamento dei componenti e riproducibilità dell'ambiente.

Tabella 1.1: *Stack* tecnologico *standard* di Spazio Dev. Fonte: `elaborazione propria`

Le tecnologie elencate nella Tabella 1.1 non sono scelte indipendenti, ma componenti di un ecosistema progettato per coprire l'intero ciclo di vita di una richiesta applicativa, come illustrato in Figura 1.1. Il nucleo di questo flusso è il *pattern* architetturale di Laravel, approfondito nella sezione seguente. A valle di esso, la risposta viene costruita tramite Blade, Livewire o Filament a seconda del tipo di interfaccia richiesta. Le operazioni che richiedono elaborazioni prolungate non bloccano la risposta sincrona in quanto Laravel Horizon gestisce una coda di *job* asincroni eseguiti da *worker* dedicati. L'intera infrastruttura è containerizzata con Docker, che isola e rende riproducibile ciascun componente. Tailwind CSS e Alpine.js completano il quadro, occupandosi della presentazione e dell'interattività lato *client*.

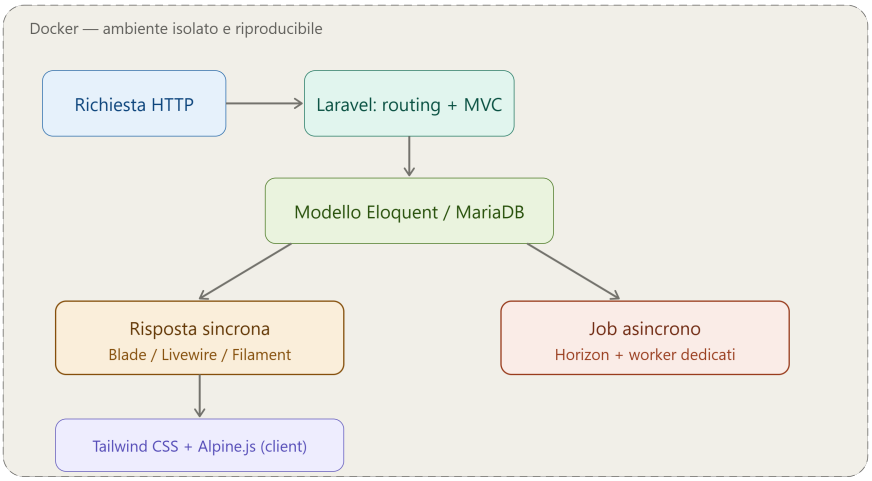

Figura 1.1: Flusso *end-to-end* di una richiesta applicativa. Fonte: `elaborazione propria`

Le sottosezioni seguenti approfondiscono singolarmente ciascuna voce della Tabella 1.1, motivandone la scelta rispetto ad altre soluzioni sul mercato.

1.2.1 PHP e Laravel

Il *backend* delle applicazioni sviluppate da Spazio Dev è realizzato in **PHP** utilizzando il *framework* **Laravel**, adottato come *standard* aziendale prima dell'inizio dello *stage*: inserendomi in progetti preesistenti, la continuità tecnologica era un requisito implicito.

Posso contestualizzare questa scelta rispetto ad alternative comuni. Il *framework* Symfony, anch'esso basato su PHP, offre maggiore flessibilità architetturale ma richiede una configurazione più articolata e presenta una curva

di apprendimento più ripida. Soluzioni basate su altri linguaggi, come Django o Express, avrebbero richiesto l'introduzione di un ecosistema tecnologico eterogeneo rispetto a quello già in uso. Laravel si distingue invece per la sua completezza: *routing*, autenticazione, gestione delle migrazioni, sistema di code e *Application Programming Interface*$_G$ (insieme di funzioni che permette a programmi diversi di comunicare e scambiarsi dati) per l'accesso al *database* sono integrati nativamente e riducono la necessità di dipendenze esterne.

L'architettura di Laravel segue il *pattern* MVC (*Model-View-Controller*$_G$) (paradigma che separa la logica applicativa, la presentazione e la gestione dei dati in tre componenti distinte, illustrato in Figura 1.2): il *router* riceve la richiesta HTTP e la instrada verso il *controller* competente, che interroga i modelli Eloquent per recuperare o modificare i dati persistiti su *database* e restituisce infine una vista (tramite Blade, descritto subito dopo) o una risposta strutturata. Questa separazione delle responsabilità è stata particolarmente utile nell'inserirmi in un progetto preesistente, poiché mi ha permesso di individuare rapidamente, per ogni nuova funzionalità da implementare, in quale livello dell'architettura intervenire.

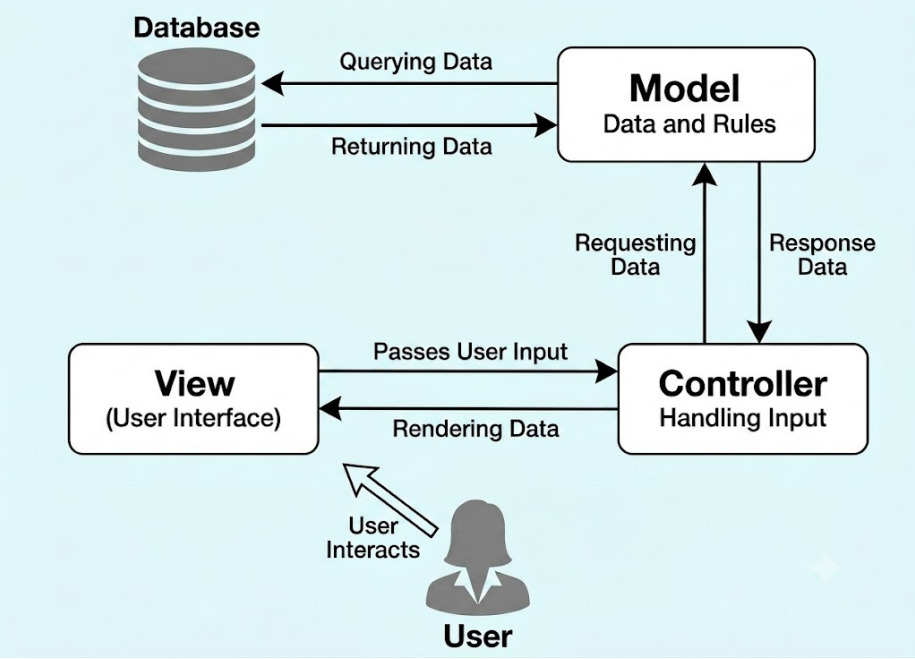

Figura 1.2: Modello *Model-View-Controller*. Fonte: `elaborazione propria`

1.2.2 Blade

Il *layer* di presentazione delle applicazioni aziendali è gestito tramite **Blade**, il motore di *template* nativo di Laravel. Blade permette di definire viste HTML con sintassi PHP *embedded*, supporta l'ereditarietà dei *layout* e favori-

sce il riutilizzo dei componenti grafici attraverso un sistema di *slot* e direttive dichiarative.

1.2.3 Livewire

Livewire è il *framework* che fornisce la base reattiva su cui è costruito Filament: permette di scrivere componenti dell'interfaccia che mantengono stato lato *server* e si aggiornano dinamicamente nel *browser* senza richiedere la scrittura manuale di *endpoint* API o codice JavaScript. Il *framework* gestisce le interazioni dell'utente tramite chiamate AJAX trasparenti, ricalcola lo stato del componente sul *server* e ne aggiorna solo le porzioni di *markup* modificate.

Insieme a Tailwind CSS, Alpine.js e Laravel, Livewire costituisce il cosiddetto *TALL stack*$_G$ (acronimo delle quattro tecnologie che lo compongono), una combinazione promossa direttamente dal *team* di Laravel come alternativa leggera, dal punto di vista dello sviluppo.

1.2.4 Filament

Per le interfacce amministrative e le sezioni CRUD delle piattaforme aziendali, Spazio Dev utilizza **Filament**, un *framework open source* che si integra nativamente con Laravel e permette di costruire pannelli di gestione dati tramite classi PHP, senza dover scrivere HTML o JavaScript per i casi d'uso *standard*.

Rispetto ad alternative come Nova, il pannello amministrativo ufficiale di Laravel, Filament presenta un modello di licenza *open source* e un ecosistema di *plugin* in rapida crescita, elementi che hanno orientato la scelta aziendale verso questa soluzione.

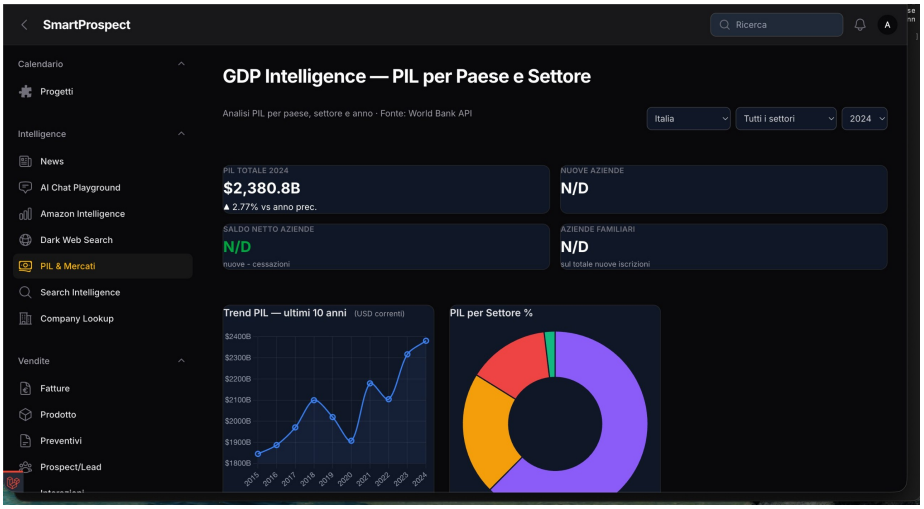

Figura 1.3: Esempio di interfaccia amministrativa generata da Filament. Fonte: `elaborazione propria`

1.2.5 Tailwind CSS e Alpine.js

Per la stilizzazione delle interfacce, l'azienda adotta **Tailwind CSS**, un *framework* CSS che segue un approccio *utility-first*: le classi di stile si applicano direttamente agli elementi HTML, eliminando la necessità di fogli di stile separati. **Alpine.js** gestisce le interazioni lato *client* tramite una sintassi dichiarativa minimale, adatta a componenti reattivi semplici senza richiedere *framework* più pesanti come React o Vue. Entrambe le tecnologie sono integrate nativamente in Filament, il che ne rende naturale l'adozione nei progetti aziendali.

1.2.6 Laravel Horizon

L'azienda affida la gestione delle operazioni asincrone a **Laravel Horizon**, uno strumento che estende il sistema di code di Laravel aggiungendo una *dashboard* di monitoraggio e funzionalità avanzate di configurazione. Horizon permette di definire quanti *worker* elaborano le code, assegnare priorità differenti a tipologie di *job* diverse e tenere traccia dei *job* falliti con i relativi messaggi di errore.

1.2.7 MariaDB

L'azienda gestisce la persistenza dei dati tramite **MariaDB**, un sistema di gestione di *database* relazionale derivato da MySQL e compatibile con esso. La scelta di un *database* relazionale rispetto a soluzioni NoSQL come MongoDB è motivata dalla natura strutturata e interconnessa dei dati trattati nei progetti aziendali: le informazioni raccolte da sorgenti diverse devono essere messe in relazione tra loro, e il modello relazionale garantisce integrità referenziale e interrogabilità tramite *join* in modo nativo. La compatibilità di MariaDB con l'ORM di Laravel, Eloquent, rende l'integrazione diretta e priva di frizioni.

1.2.8 Docker

L'azienda containerizza l'infrastruttura applicativa con **Docker**, che consente di isolare ciascun componente di un sistema in *container* indipendenti, garantendo riproducibilità dell'ambiente e separazione delle responsabilità. Questo isolamento è particolarmente rilevante nei sistemi distribuiti, dove i diversi componenti hanno cicli di vita e requisiti di risorse differenti.

La Figura 1.4 illustra il principio alla base di questa scelta, confrontando un monolite con un sistema containerizzato: in un'applicazione non containerizzata, processi diversi (*server web*, *worker* delle code, *scheduler*) condividono lo stesso

ambiente di esecuzione, per cui un aggiornamento o un guasto su uno di essi può ripercuotersi sugli altri. Containerizzando ciascun processo separatamente, ogni componente può essere aggiornato, riavviato o sottoposto a *debug* in isolamento. È importante notare che questo non equivale di per sé a un'architettura a microservizi: anche un'applicazione monolitica, che condivide un'unica *codebase* e un unico *database*, può beneficiare di questo isolamento eseguendo i propri processi in *container* distinti, come avviene per il progetto SmartProspect.

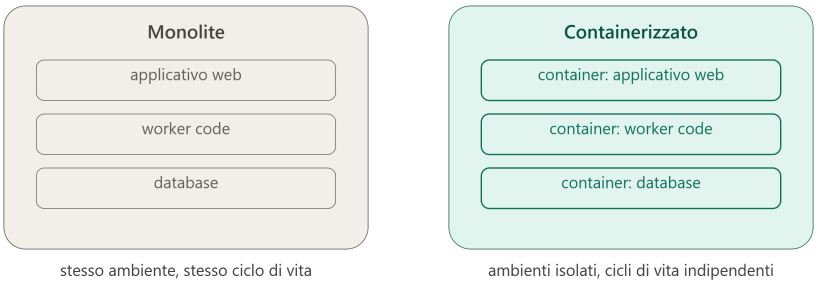

Figura 1.4: Isolamento dei processi tramite containerizzazione. Fonte: `elaborazione propria`

Nei progetti osservati, ciascun *container* corrisponde tipicamente a un singolo processo o a una singola responsabilità infrastrutturale: l'applicativo Laravel, il *database* MariaDB, il *worker* delle code gestito da Horizon. Questa granularità consente di aggiornare o riavviare un singolo componente senza impattare gli altri, un vantaggio concreto in un sistema in cui le diverse parti hanno cicli di vita e frequenze di rilascio differenti.

1.3 Processo di sviluppo del prodotto

Il processo di sviluppo osservato in azienda si concentra prevalentemente sulle attività di progettazione, implementazione e verifica del prodotto, essendo SmartProspect una piattaforma già parzialmente esistente e in fase di consolidamento. Non ho quindi osservato momenti formali di analisi di fattibilità o di pianificazione commerciale, generalmente collocati nelle attività preliminari alla realizzazione di un nuovo prodotto. L'attività si è concentrata sul ciclo breve e iterativo che porta dalla definizione di una funzionalità alla sua integrazione stabile nel sistema in produzione.

Il versionamento del codice nei progetti aziendali è gestito tramite Gitea, una piattaforma di *hosting* Git *self-hosted*. Lo sviluppo avviene su *branch* dedicati separati dal *branch* di produzione già rilasciato: al termine di ciascuna attività

si apre una *pull request* verso il *branch* principale, revisionata e integrata dal *team* aziendale. Questa separazione consente di lavorare in modo continuativo senza rischiare di compromettere gli ambienti di produzione.

Una pratica centrale del processo di sviluppo aziendale è la *code review* quotidiana condotta dal *tutor*, che permette di mantenere la coerenza del codice con gli *standard* adottati e rappresenta un importante strumento di apprendimento per le nuove risorse.

Il *team* ridefinisce le attività settimana per settimana in funzione dell'avanzamento raggiunto e delle criticità emerse, mantenendo un confronto costante tra le risorse coinvolte nel progetto e il *tutor* aziendale. Ogni ciclo si apre con una sessione di pianificazione delle attività e si chiude con una verifica del lavoro prodotto, accompagnata da *feedback* tecnico e indicazioni per il ciclo successivo.

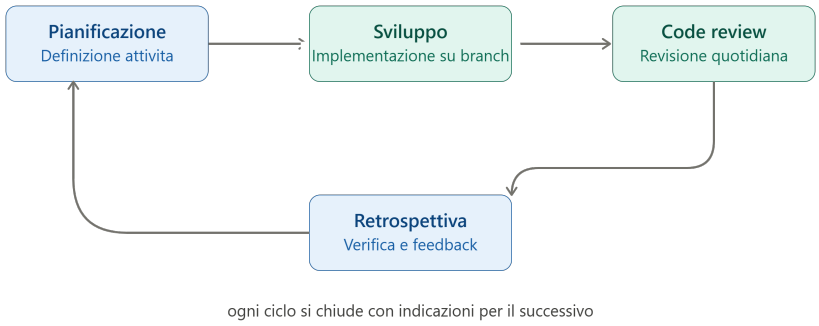

Figura 1.5: Ciclo settimanale di sviluppo adottato nel progetto. Fonte: `elaborazione propria`

Come illustrato in Figura 1.5, la fase di *code review* non rappresenta un controllo isolato a fine settimana, ma un processo continuo che accompagna lo sviluppo durante tutto l'arco della settimana. Ogni *pull request* veniva revisionata prima della retrospettiva, in modo che le criticità tecniche fossero già risolte al momento del confronto con il *tutor*, lasciando alla retrospettiva il compito di discutere priorità e organizzazione del ciclo successivo.

La mia partecipazione agli incontri di allineamento del *team* è stata occasionale più che sistematica, poiché non ho preso parte a ogni singolo *stand-up*, ma solo a quelli più rilevanti per l'avanzamento delle attività in corso. Un confronto conclusivo con l'intero *staff* aziendale si è invece tenuto al termine dello *stage*, dedicato alla presentazione e alla valutazione del prodotto sviluppato nel corso dei due mesi.

1.4 Propensione all'innovazione

Diversi elementi osservati durante lo *stage* indicano un orientamento aziendale verso tecnologie recenti piuttosto che verso soluzioni consolidate ma più rigide, come l'adozione di Filament o l'integrazione di strumenti di intelligenza artificiale generativa all'interno delle piattaforme sviluppate.

Lavorando direttamente sui moduli di IA del progetto, ho potuto osservare che questa propensione non si esaurisce nella scelta degli strumenti, bensì si riflette anche nella disponibilità a destinare tempo di sviluppo a funzionalità ancora sperimentali, il cui valore concreto per il prodotto non era del tutto certo a priori. È una propensione all'innovazione che si misura tanto nelle tecnologie adottate quanto nella tolleranza al rischio con cui l'azienda le introduce.

Un ulteriore indicatore di questa propensione è la prassi, consolidata da diversi anni, di ospitare stagisti universitari. L'azienda integra regolarmente queste figure nel proprio organico al termine del percorso, segno di una strategia di selezione del personale che privilegia la formazione interna rispetto al reclutamento esterno. Approfondirò questo aspetto e il modo in cui il presente tirocinio si inserisce in tale strategia nel Capitolo 2.

Capitolo 2

Le ragioni del tirocinio

Questo Capitolo spiega perché Spazio Dev ha proposto questo specifico tirocinio: il rapporto dell'azienda con l'ospitalità di stagisti, il problema aziendale che il tirocinio doveva contribuire ad affrontare, gli obiettivi e i vincoli concordati e le ragioni per cui ho preferito questa proposta ad altre.

2.1 L'ospitalità dei tirocini in azienda

Come accennato nel Capitolo 1, Spazio Dev ospita tirocinanti in modo continuativo. Dal 2024 mantiene costantemente tre posizioni di *stage* universitario occupate, con un meccanismo di *turnover* diretto: quando uno stagista conclude il proprio percorso, l'azienda copre immediatamente la posizione con una nuova figura. Non si tratta quindi di un'iniziativa occasionale legata a singole esigenze di progetto, ma di una componente stabile dell'organico aziendale.

Oltre al canale universitario, l'azienda accoglie anche studenti delle scuole superiori per percorsi di alternanza scuola-lavoro, con finalità prevalentemente didattiche. La Tabella 2.2 riassume le differenze principali tra i due canali, per come li ho osservati durante lo *stage*.

	***Stage* universitario**	**Alternanza scuola-lavoro**
Finalità principale	Selezione del personale e contributo produttivo	Orientamento e apprendimento di base
Provenienza	Quasi esclusivamente dal Corso di Laurea in Informatica dell'Università di Padova	Istituti scolastici locali
Posizioni occupate	Costantemente 3, con *turnover* diretto	Variabile, non continuativo

Tabella 2.2: Confronto tra i due canali di tirocinio osservati in azienda durante lo *stage*. Fonte: `elaborazione propria`

Il percorso tipico dello stagista universitario segue uno schema ricorrente: un periodo iniziale di affiancamento al *tutor*, seguito rapidamente da un'autonomia crescente nello svolgimento delle attività. Le richieste operative possono variare anche in modo significativo *in itinere*, in funzione delle priorità aziendali del momento, un aspetto che ho potuto verificare direttamente nella quarta settimana del tirocinio (Capitolo 3), quando un'attività trasversale legata a un caso d'uso reale ha temporaneamente rimodellato il lavoro pianificato. Al termine del percorso, la prosecuzione del rapporto tramite assunzione dipende dalla compatibilità reciproca tra le esigenze dell'azienda e quelle dello stagista.

Questo meccanismo di *turnover* costante e l'investimento richiesto per affiancare e formare nuove risorse, anche quando non porta necessariamente a un'assunzione, rappresentano un ulteriore indicatore dell'inclinazione verso l'innovazione di Spazio Dev discussa nel Capitolo 1: l'azienda sceglie di costruire competenze internamente, anche a fronte di una rotazione del personale che impone un costo di formazione ricorrente, invece di limitarsi al solo reclutamento di personale già formato sul mercato.

2.2 Il problema aziendale dello *stage*

La difficoltà di raccogliere e mettere a sistema informazioni disperse tra fonti eterogenee non è una criticità isolata di Spazio Dev, ma riflette una tendenza più ampia osservabile nel mondo del lavoro contemporaneo: la quantità di dati potenzialmente rilevanti per una decisione aziendale, distribuiti tra siti *web*, registri pubblici, motori di ricerca, documenti interni, cresce più rapidamente

della capacità umana di raccoglierli e interpretarli manualmente. L'automazione di questi processi di raccolta e normalizzazione non è un'esigenza specifica di un singolo settore, ma una necessità trasversale a qualunque organizzazione che voglia basare le proprie decisioni su informazioni aggiornate e verificabili.

Il tirocinio si inserisce nello sviluppo di SmartProspect, una piattaforma *software* orientata alla raccolta, gestione e analisi di dati aziendali provenienti da sorgenti eterogenee, già parzialmente esistente al momento dell'inizio dello *stage*. Il problema che l'azienda intendeva affrontare riguarda proprio questa difficoltà, per le proprie attività commerciali, di raccogliere e mettere a sistema informazioni disperse tra fonti diverse a supporto di processi decisionali e di *business intelligence*.

L'attività di *stage* mirava all'estensione e al consolidamento delle funzionalità già presenti nell'ecosistema applicativo, con particolare attenzione ai processi di acquisizione automatizzata dei dati, alla loro strutturazione all'interno del sistema informativo nonché alla predisposizione di strumenti per l'analisi e la consultazione.

2.3 Obiettivi e vincoli dello *stage*

Le attività di *stage* sono state pianificate su un arco di otto settimane per un totale di circa 300 ore, con una distribuzione di 40 ore settimanali. Descriverò nel dettaglio la pianificazione effettiva delle attività nel Capitolo 3.

Obiettivi aziendali

Gli obiettivi tecnici dello *stage* derivano dal piano di lavoro concordato con l'azienda, articolato in cinque fasi, ciascuna con una durata stimata e un risultato atteso verificabile:

1. **Sviluppo dei *job* di orchestrazione** per una durata di tre settimane: realizzare *job* compatibili con Laravel Horizon capaci di inviare e ricevere comandi verso e dai *container* di *scraping* presenti nell'ecosistema applicativo, rendendo l'integrazione esistente più stabile e operativa su almeno tre sorgenti dati distinte, quali Google Maps, OpenStreetMap e fonti *web* generiche.

2. **Persistenza dei dati raccolti** per la durata di una settimana: implementare la *pipeline* di trasferimento dei dati acquisiti dagli *scraper* verso il *database*, garantendo che i *record* siano correttamente strutturati.

3. **Normalizzazione e pulizia dei dati** per la durata di una settimana: sviluppare i processi di pulizia e normalizzazione necessari a rendere i dati uniformi e privi di anomalie o duplicati, così da renderli affidabili per l'analisi.

4. **Interfaccia *web* di visualizzazione** per una durata di due settimane: realizzare le viste Filament che permettano all'utente di consultare i dati raccolti, avvalendosi di filtri.

5. ***Report* finale e presentazione al *team*** per la durata di una settimana: produrre la documentazione del lavoro svolto e presentarne i risultati al *team* aziendale, evidenziando le relazioni tra i dati raccolti e le funzionalità implementate.

Complessivamente, l'obiettivo dell'azienda era dare vita a una piattaforma completa in grado di estrapolare in autonomia informazioni pubblicamente disponibili riguardanti aziende *competitor*. La piattaforma può essere considerata uno strumento dalla natura polivalente, poiché genera valore su più fronti. Da un lato, l'azienda stessa ne trae beneficio diretto raccogliendo informazioni rilevanti da confrontare con la propria attività; dall'altro, può rappresentare un utile strumento di analisi per clienti che richiedono consulenze di *marketing* a supporto del proprio *business*.

KPI	Valore	Interpretazione
Importo totale speso	33.986,93 EUR	Spesa sufficiente per individuare *benchmark* storici e riallocare in modo *data driven*.
Impression totali	14.943.938	Capillarità elevata nel bacino locale e nelle campagne storiche.
Reach totale	3.022.289	Buona ampiezza di contatto, da bilanciare con qualità e frequenza.
Frequenza media	4,94	Segnale di pressione pubblicitaria; richiede rotazione creativa.
CPM medio	2,27 EUR	Costo misto accettabile, ma migliorabile con *awareness* efficiente.
CPM *high-efficiency awareness*	0,54 EUR	*Benchmark* per costruire bacini di *retargeting* a costo basso.

Tabella 2.4: Esempio di *report* KPI *macro advertising* estratto tramite SmartProspect per un cliente reale, con interpretazione dei valori a supporto delle decisioni di *marketing*. Fonte: `elaborazione propria`

Obiettivi personali

A livello personale, gli obiettivi che mi ero posta all'avvio del tirocinio erano più generali rispetto a quelli tecnici concordati con l'azienda:

- acquisire una comprensione operativa di un *framework* completo come Laravel, toccando con mano più aree del suo ecosistema e non specializzarmi fin da subito su un singolo aspetto;
- lavorare su problemi di natura diversa nel corso del tirocinio, come integrazione di sistemi esterni e automazione, per misurare la mia capacità di adattamento a contesti tecnici non omogenei;
- imparare a muovermi all'interno di una *codebase* preesistente e non realizzata dalla sottoscritta, comprendendone le convenzioni prima di introdurre

nuove funzionalità. Questa è una competenza distinta da quella richiesta nello scrivere codice da zero e più rappresentativa del lavoro su progetti reali;

- abituarmi a un metodo di lavoro strutturato attorno a cicli di pianificazione, revisione e *feedback* costante con un *tutor*, imparando ad adattare le priorità settimana per settimana in funzione dell'avanzamento reale;
- maturare una valutazione informata, basata sull'esperienza diretta, su quanto il lavoro di sviluppo *software* in un contesto aziendale corrisponda alle mie aspettative e ai miei interessi professionali.

Nel Capitolo 4 discuto il bilancio rispetto a entrambe le categorie di obiettivi descritte.

2.4 Analisi preventiva dei rischi

Durante l'analisi iniziale ho individuato i principali rischi a cui il progetto poteva essere soggetto. Per ciascun rischio ho definito strategie di mitigazione adottate nel corso dello sviluppo. Ho suddiviso i rischi in due categorie: tecnici, legati alla natura del sistema sviluppato, e organizzativi, legati alla gestione del lavoro e al contesto di *stage*.

2.4.1 Rischi tecnici

Descrizione: le sorgenti dati integrate nel progetto potrebbero restituire dati incompleti, non aggiornati o strutturalmente inconsistenti, rendendo difficile la normalizzazione e l'analisi.
Mitigazione: introduzione di processi di validazione e trasformazione intermedi, con gestione esplicita dei casi di errore e dato mancante.

Rischio verificato: i meccanismi di fallback e validazione introdotti si sono rivelati necessari per mantenere la coerenza dei dati persistiti nel sistema.

Descrizione: un sistema basato su code di *job*, *container* isolati e sorgenti esterne introduce complessità nella tracciabilità degli errori e nella gestione degli stati di esecuzione, rendendo il *debug* intrinsecamente più difficile che in un'applicazione monolitica.
Mitigazione: suddivisione delle operazioni in *job* atomici, adozione di *logging* strutturato e monitoraggio tramite Laravel Horizon.

Rischio verificato: il logging e la dashboard di Horizon si sono rivelati indispensabili per circoscrivere rapidamente l'origine degli errori nell'integrazione tra componenti.

Descrizione: la dipendenza da servizi e *container* esterni potrebbe introdurre latenza, interruzioni o cali di prestazioni durante elaborazioni massive, con impatto sulla stabilità complessiva del sistema.
Mitigazione: isolamento delle componenti tramite interfacce standardizzate e architettura asincrona con separazione tra fase di acquisizione, elaborazione e consultazione.

Rischio verificato: l'isolamento ha permesso di contenere l'impatto di malfunzionamenti localizzati senza propagarli all'intero sistema.

2.4.2 Rischi organizzativi

Descrizione: l'intero *stack* tecnologico, da Laravel a Filament e a Horizon, era nuovo al momento dell'inizio del tirocinio. Il tempo necessario ad acquisire una padronanza sufficiente per contribuire efficacemente al progetto potrebbe essere sottostimato.
Mitigazione: utilizzo della prima settimana per la formazione e l'analisi del codice esistente, con supporto diretto del *tutor* aziendale.

Questo rischio non si è concretizzato: la formazione iniziale prevista nella prima settimana si è rivelata sufficiente ad acquisire l'autonomia necessaria, senza che la curva di apprendimento incidesse in modo significativo sui tempi delle fasi successive.

Descrizione: inserirsi in un progetto già parzialmente sviluppato comporta il rischio di non comprendere appieno le scelte architetturali già effettuate, con conseguente produzione di codice incoerente rispetto agli *standard* del *team* o necessità di rielaborare quanto realizzato.
Mitigazione: analisi approfondita del codice esistente prima di introdurre nuove funzionalità e confronto sistematico con il *tutor* in caso di dubbi interpretativi.

Anche questo rischio non si è mai concretizzato: l'analisi preventiva del codice esistente e il confronto sistematico con il tutor si sono dimostrati sufficienti a evitare fraintendimenti significativi sull'architettura preesistente.

Descrizione: la pianificazione settimanale potrebbe essere invalidata da blocchi tecnici imprevisti, da una stima errata della durata delle attività o da richieste di modifica emerse in corso d'opera, con il rischio di accumulare ritardi difficili da recuperare nelle fasi successive.
Mitigazione: revisione esplicita delle priorità a ogni incontro settimanale con il *tutor*, con ridefinizione delle *task* in funzione dell'avanzamento effettivo piuttosto che di quello atteso.

Questo è l'unico rischio organizzativo che si è effettivamente concretizzato, durante la quarta settimana, quando il coinvolgimento temporaneo in un'attività parallela ha causato un rallentamento rispetto al piano originale. Tuttavia, la mitigazione prevista si è rivelata sufficiente ad assorbire il ritardo senza comprometterne il recupero nelle settimane successive.

Descrizione: durante le attività di sviluppo autonomo, il rischio di rimanere bloccati su un problema tecnico per un tempo eccessivo può rallentare significativamente l'avanzamento del progetto.
Mitigazione: definizione di una soglia temporale oltre la quale escalare il problema al *tutor*, evitando sia la dipendenza eccessiva dal supporto esterno sia la perdita di tempo in tentativi infruttuosi.

Questo rischio non si è mai concretizzato nel corso dello stage perché non si sono verificati episodi di blocco prolungato su un singolo problema.

2.5 Perché questo *stage*

Ho conosciuto Spazio Dev in occasione del Career Day dell'Università di Padova, nel marzo 2026, dove l'azienda presentava SmartProspect, insieme ad altri progetti, e gli ambiti di sviluppo a esso collegati. Tra le proposte di tirocinio che ho preso in considerazione, questa è quella che mi ha colpita maggiormente per l'orientamento del progetto verso la *business intelligence* e l'analisi dati: un ambito applicativo che univa sviluppo *software* a un dominio che trovavo più stimolante rispetto a proposte orientate a sviluppo puramente gestionale o a manutenzione di sistemi esistenti senza una componente analitica.

Un fattore aggiuntivo nella scelta è stata la possibilità di lavorare su uno *stack* tecnologico che, pur non conoscendo in precedenza, presentava un buon equilibrio tra solidità del *framework* e rilevanza nel mercato del lavoro per sviluppatori *full-stack*, rendendolo una competenza che ritenevo valesse la pena acquisire.

La sede dell'azienda, a Tombolo (PD), comportava un impegno logistico non trascurabile rispetto ad altre opzioni più vicine alla mia residenza, ma ho comunque ritenuto che l'interesse per il progetto giustificasse questo aspetto organizzativo.

Capitolo 3

Il progetto SmartProspect

Figura 3.1: Logo SmartProspect. Fonte: `elaborazione propria`

Questo Capitolo racconta il progetto oggetto di *stage* dall'idea alla sua realizzazione: il metodo di lavoro adottato, i problemi progettuali, tecnologici e applicativi affrontati seguendo la sequenza analisi -> progettazione -> programmazione -> verifica e i risultati raggiunti, sia sul piano qualitativo che quantitativo.

3.1 Metodo di lavoro

3.1.1 Pianificazione e revisione del lavoro

La pianificazione iniziale ha subito alcuni adattamenti nel corso dello sviluppo: il ritmo di lavoro è stato generalmente incalzante e mi ha portata ad anticipare diverse attività rispetto alla calendarizzazione originaria, in funzione dell'avanzamento effettivo del progetto. Come già accennato, una delle settimane ha registrato un lieve rallentamento a causa del coinvolgimento in una *task*

trasversale rispetto alle attività previste dal piano di lavoro. Nel complesso, tuttavia, l'avanzamento è rimasto in linea con gli obiettivi dello *stage*.

Di seguito descrivo la distribuzione effettiva delle attività per ciascuna settimana.

Settimana 1

Ho dedicato i primi giorni alla configurazione dell'ambiente di sviluppo e allo studio delle tecnologie e dei *framework* adottati dall'azienda (descritti nel Capitolo 1). Questa fase ha incluso l'analisi dell'architettura esistente e la comprensione del funzionamento dei moduli già presenti, necessarie prima di poter contribuire attivamente con nuove funzionalità. Già durante questa settimana, alcune attività sono state anticipate rispetto alla pianificazione originale, consentendomi di entrare fin da subito nel flusso di lavoro e acquisire familiarità con gli strumenti utilizzati.

Settimana 2

Nella seconda settimana ho avviato l'integrazione con il *database* MariaDB, organizzando i dati raccolti in tabelle distinte e definendo le relazioni necessarie tra di esse. Parallelamente ho sviluppato le prime viste CRUD all'interno dell'interfaccia amministrativa, introducendo funzionalità di visualizzazione tabellare, inserimento manuale dei *record* e importazione massiva dei dati tramite *file* CSV.

Settimana 3

Nel corso della terza settimana ho cominciato le attività di *scraping* finalizzate alla raccolta dei dati *target*. In contemporanea è iniziata la progettazione della sezione di *intelligence*, orientata alla generazione di *overview* e *insight* a partire dall'analisi incrociata delle informazioni raccolte. Anche in questa settimana alcune attività sono state anticipate rispetto alla pianificazione originale.

Settimana 4

La quarta settimana ha previsto un coinvolgimento temporaneo in un'attività concomitante, finalizzata alla realizzazione di un'analisi per un cliente specifico. Il contesto applicativo faceva uso delle stesse tecnologie e seguiva il medesimo flusso di lavoro della piattaforma principale, ma era orientato allo studio dei dati e alla produzione di reportistica. L'attività ha incluso momen-

ti di confronto diretto con gli *stakeholder* e il supporto alla presentazione dei risultati emersi.

In parallelo, ho consolidato gli ultimi *job* e i processi di orchestrazione e raccolta dati, oltre ad aver risolto alcune problematiche emerse durante l'integrazione delle componenti. L'avanzamento ha subito un lieve rallentamento rispetto alle settimane precedenti, senza tuttavia compromettere il raggiungimento degli obiettivi previsti per lo *stage*.

Settimana 5

Durante la quinta settimana ho sviluppato due nuovi moduli di acquisizione dati. Il primo consiste in un *crawler* dedicato alla raccolta periodica di informazioni da pagine pubbliche di Amazon, con l'obiettivo di monitorarne l'andamento nel tempo e conservarne uno storico. Il secondo riguarda invece la raccolta di dati macroeconomici destinati alla produzione di *report* e analisi del tessuto economico. Parallelamente ho risolto alcune anomalie emerse nell'ambiente di produzione.

Settimana 6

Durante la sesta settimana ho sviluppato due componenti basate su intelligenza artificiale per ampliare le capacità analitiche della piattaforma. La prima consiste nell'integrazione di un agente IA in grado di raccogliere dati testuali dal *web* e organizzarli in un *database* dedicato, interrogabile tramite un'interfaccia conversazionale in linguaggio naturale. La seconda riguarda l'integrazione con Google Drive tramite un sistema di tipo *Retrieval-Augmented Generation*$_G$ (tecnica usata nell'intelligenza artificiale per far sì che un modello generativo risponda usando anche documenti esterni), che consente di collegare un *account*, selezionare una cartella e indicizzare automaticamente i documenti PDF per successive interrogazioni contestuali. A fine settimana l'applicativo risultava quasi completo nelle sue funzionalità principali.

Settimana 7

Nel corso della settimana ho svolto attività di consolidamento delle funzionalità sviluppate nelle settimane precedenti, accompagnate da una verifica del corretto funzionamento della piattaforma. Ho effettuato ulteriori controlli sulle componenti di raccolta e consultazione dei dati, introducendo alcune migliorie grafiche volte a rendere l'esperienza utente più fluida. Ho inoltre avviato la

redazione della documentazione relativa al progetto, in vista della conclusione del tirocinio.

Settimana 8

L'ultima settimana è stata dedicata alla finalizzazione della documentazione di progetto e alla chiusura delle attività di tirocinio. Ho partecipato agli incontri conclusivi di allineamento con il *team* aziendale, durante i quali abbiamo discusso i risultati raggiunti nel corso dei mesi di *stage* e ho raccolto i relativi *feedback*. Con la conclusione di questa fase si è chiuso il ciclo di sviluppo previsto dal progetto.

La tabella seguente riassume la distribuzione delle ore per fase, secondo quanto previsto dalla pianificazione iniziale concordata con il *tutor* aziendale.

Ore	**Attività**
40	Formazione sulle tecnologie e analisi architettura esistente
80	Configurazione, prototipazione e prime funzionalità
40	Sviluppo *core* e integrazione *database*
40	Avanzamento sviluppo e *test* intermedi
80	Analisi e visualizzazione dati
20	Presentazione, documentazione e conclusione
300	**Totale**

Tabella 3.1: Distribuzione delle ore per fase di lavoro

3.1.2 Metodologia *Agile* e strumenti di tracciamento

Lo sviluppo dell'applicativo SmartProspect ha seguito i principi della metodologia *Agile*$_G$: il lavoro è stato organizzato in cicli iterativi (*sprint*) della durata approssimativa di due settimane. Ciascuna iterazione si apriva con un *briefing* iniziale durante il quale si definiva il perimetro dei moduli da implementare. Al termine dello *sprint*, l'applicativo veniva sottoposto a sessioni di revisione e retrospettiva per analizzare le criticità riscontrate e pianificare i passi successivi. Il tracciamento e la gestione delle *issue* sono stati centralizzati mediante la piattaforma *Plane*.

Un fattore determinante nel ciclo di vita del *software* è stato l'adattamento continuo ai vincoli imposti dagli ambienti esterni. Trattandosi di un sistema fortemente incentrato sulle automazioni, i dati estratti dal *web* si presentavano frequentemente incompleti, non strutturati o del tutto assenti, introducendo il

rischio di *bug* a catena e fallimenti nei *job* di persistenza. Il ciclo di sviluppo ha quindi previsto un *continuous refactoring*, focalizzato sull'introduzione di complesse cascate di *fallback* e logiche di *rate-limiting* stringenti, necessarie a garantire la resilienza del sistema a fronte di blocchi IP o variazioni strutturali dei siti *target*.

3.2 Analisi dei requisiti

3.2.1 Casi d'uso

L'analisi dei requisiti è partita dalla modellazione dei casi d'uso dell'intera piattaforma. Il metodo seguito è stato di tipo *top-down*: ho dapprima identificato, insieme al *tutor*, le macro-aree funzionali visibili all'attore principale del sistema, l'**Utente** autenticato, e ho associato a ciascuna di esse un caso d'uso principale. Ho poi decomposto ogni caso d'uso principale nelle sue varianti, adottando la convenzione di numerazione gerarchica `UC-x.y`: il livello `x` identifica la macro-funzionalità, il livello `y` le sotto-funzionalità raggiungibili da essa, collegate tramite relazioni formali di associazione, inclusione (`«include»`, quando la delega è obbligatoria) o estensione (`«extends»`, per comportamenti opzionali o eccezionali). Ogni caso d'uso è corredato da una specifica testuale e da un diagramma *Unified Modeling Language*$_G$ (un linguaggio di modellazione grafico *standard* per la progettazione orientata agli oggetti).

La descrizione testuale di ciascun caso d'uso è strutturata secondo le seguenti informazioni:

- **Attori Principali**: rappresentano i ruoli esterni (umani o sistemi terzi) che avviano l'interazione o ne traggono il valore principale.
- **Precondizioni**: lo stato in cui il sistema deve necessariamente trovarsi affinché il caso d'uso possa essere avviato.
- **Postcondizioni**: i vincoli e le modifiche strutturali che permangono nello stato del sistema.
- **Scenari Alternativi**: flussi secondari che descrivono deviazioni dal percorso principale, tipicamente associati a fallimenti della validazione dei dati, eccezioni tecniche o interruzioni del processo.

Complessivamente la modellazione ha prodotto **11 casi d'uso principali** (dall'autenticazione UC-0 alla SEO Intelligence UC-10), articolati in varianti per un totale di **36 identificatori UC**. In questa sede presento nel dettaglio

i due casi d'uso che ritengo più interessanti: **UC-7** (*knowledge base* RAG) e **UC-8** (Research Agent). La scelta ricade su questi due per tre ragioni: sono i moduli principali che ho progettato e sviluppato interamente durante lo *stage*, esemplificano bene la decomposizione gerarchica adottata e sono gli stessi sottosistemi ripresi nelle sezioni di progettazione (§3.3.4, §3.3.5) e di verifica (§3.4.4), consentendo al lettore di seguirne il ciclo di vita completo, dall'analisi al collaudo.

UC7: Gestione *knowledge base* RAG

Attori Principali: Utente.
Precondizioni: L'utente è autenticato nel sistema.
Descrizione: L'utente accede al cruscotto RAG. Il sistema verifica lo stato della connessione con Google Drive:

1. Se l'*account* **non è connesso**, il sistema richiede l'autenticazione includendo il flusso dedicato (UC-7.1).

2. Se l'*account* **è connesso**, il sistema visualizza lo stato corrente della *knowledge base* (`syncing` o `ready`, con il numero di *file* indicizzati).

Da questo punto l'utente può scegliere se navigare il proprio Drive per selezionare e sincronizzare nuovi documenti (UC-7.2) oppure avviare una sessione di *chat* RAG (UC-7.3).
Postcondizioni: L'utente visualizza lo stato aggiornato della *knowledge base* e ha accesso alle sotto-funzionalità di sincronizzazione e *chat*.
Scenario Alternativo: Google Drive non è connesso: il sistema mostra lo stato *non connesso* e attiva il flusso di autenticazione OAuth2 (vedi UC-7.1).

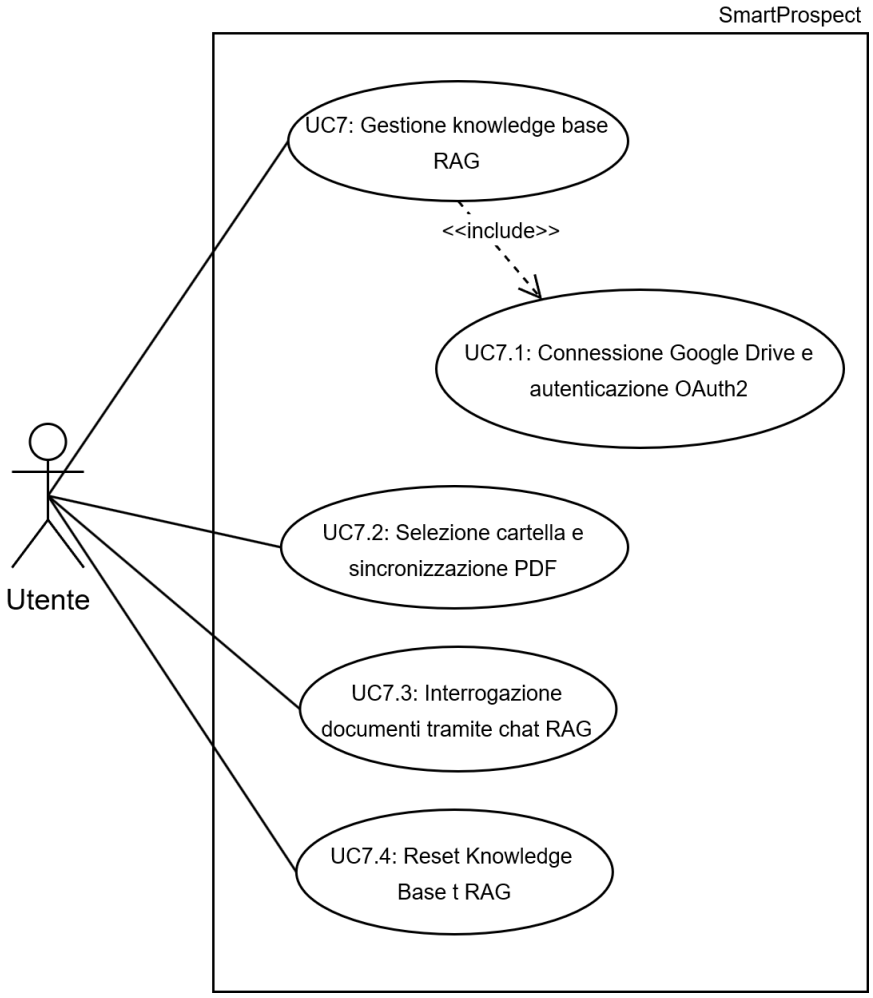

Figura 3.2: Diagramma del caso d'uso UC-7

UC7.1: Connessione Google Drive e autenticazione OAuth2

Attori Principali: Utente.
Precondizioni: L'utente si trova nel cruscotto RAG (UC-7) e l'*account* Google Drive non risulta associato.
Descrizione: L'utente avvia il flusso di autenticazione OAuth2 tramite il pulsante di connessione. Il sistema reindirizza l'utente alla pagina di consenso ufficiale di Google. Una volta concesso il consenso, il sistema acquisisce i *token* di accesso e di *refresh*. Al termine, il sistema aggiorna lo stato della connessione nell'interfaccia utente, sbloccando le funzionalità dipendenti.
Postcondizioni: La connessione Google Drive è attiva.
Scenario Alternativo: L'utente nega il consenso o si verifica un errore OAuth2: il sistema notifica l'errore tramite un messaggio in interfaccia e mantiene lo stato *non connesso*.

UC7.2: Selezione cartella e sincronizzazione PDF

Attori Principali: Utente.
Precondizioni: L'utente si trova nel cruscotto RAG (UC-7) e la connessione con Google Drive è attiva (vedi UC-7.1).
Descrizione: L'utente esplora la gerarchia delle cartelle del proprio Drive remoto attraverso l'interfaccia del *browser* a due fasi. Nella fase di *browse*, l'utente naviga fino a selezionare la cartella desiderata; nella fase di *confirm*, il sistema scansiona la cartella, rileva i PDF presenti e li mostra pre-selezionati in un elenco, consentendo all'utente di rifinire la scelta.
Postcondizioni: I *file* PDF selezionati sono indicizzati nel *vector store* di OpenAI.
Scenario Alternativo: Il *download* di uno o più PDF fallisce: il sistema salta i *file* corrotti, garantisce la rimozione sicura dei *file* temporanei dal *server* e aggiorna il contatore includendo solo i documenti elaborati con successo.

UC7.3: Interrogazione documenti tramite *chat* RAG

Attori Principali: Utente.
Precondizioni: L'utente si trova nel cruscotto RAG (UC-7) e la *knowledge base* è in stato `ready` con almeno un documento indicizzato.
Descrizione: L'utente inserisce un'interrogazione in linguaggio naturale nella *chat*. Il sistema memorizza istantaneamente il messaggio dell'utente nel *database* locale.
Postcondizioni: La risposta dell'agente RAG è mostrata a schermo e salvata nella cronologia della conversazione; ogni asserzione è tracciabile direttamente al documento sorgente.
Scenario Alternativo: Nessun documento contiene informazioni pertinenti alla domanda: l'agente RAG rileva la mancanza di contesto utile nel *vector store* e risponde esplicitamente notificando all'utente l'assenza di informazioni rilevanti, evitando di generare allucinazioni.

UC7.4: *Reset knowledge base* RAG

Attori Principali: Utente.
Precondizioni: L'utente si trova nel cruscotto RAG (UC-7) e la *knowledge base* contiene almeno un documento indicizzato o messaggi di conversazione salvati.
Descrizione: L'utente richiede la cancellazione completa della *knowledge base* tramite l'apposita azione di interfaccia. Il sistema richiede una conferma espli-

cita prima di procedere; una volta confermata, il sistema elimina il *vector store* su OpenAI, tutti i metadati associati e la cronologia delle conversazioni locali. La *dashboard* viene ricaricata allo stato iniziale.
Postcondizioni: La *knowledge base* è azzerata; il cruscotto RAG si presenta nello stato iniziale con il contatore dei documenti a 0 e la cronologia delle conversazioni vuota.
Scenario Alternativo: L'eliminazione del *vector store* fallisce a causa di un errore nelle API OpenAI: il sistema notifica il fallimento tramite un messaggio di errore e mantiene intatti i dati esistenti.

UC8: Ricerca autonoma sul *web* con Research Agent

Attori Principali: Utente.
Precondizioni: L'utente è autenticato nel sistema.
Descrizione: L'utente accede alla *dashboard* del Research Agent. Il sistema visualizza il numero di documenti già indicizzati nella sessione corrente e la cronologia delle conversazioni passate. Da questo punto l'utente può scegliere di avviare una nuova *pipeline* di ricerca autonoma sul *web* (UC-8.1), effettuare un'interrogazione analitica sui soli documenti già raccolti (UC-8.2), oppure azzerare completamente lo stato della sessione (UC-8.3).
Postcondizioni: L'utente visualizza la *dashboard* del Research Agent aggiornata e abilitata all'interazione.
Scenario Alternativo: Nessuna sessione precedente è attiva: il sistema inizializza una nuova sessione vuota, mostrando il contatore dei documenti a 0.

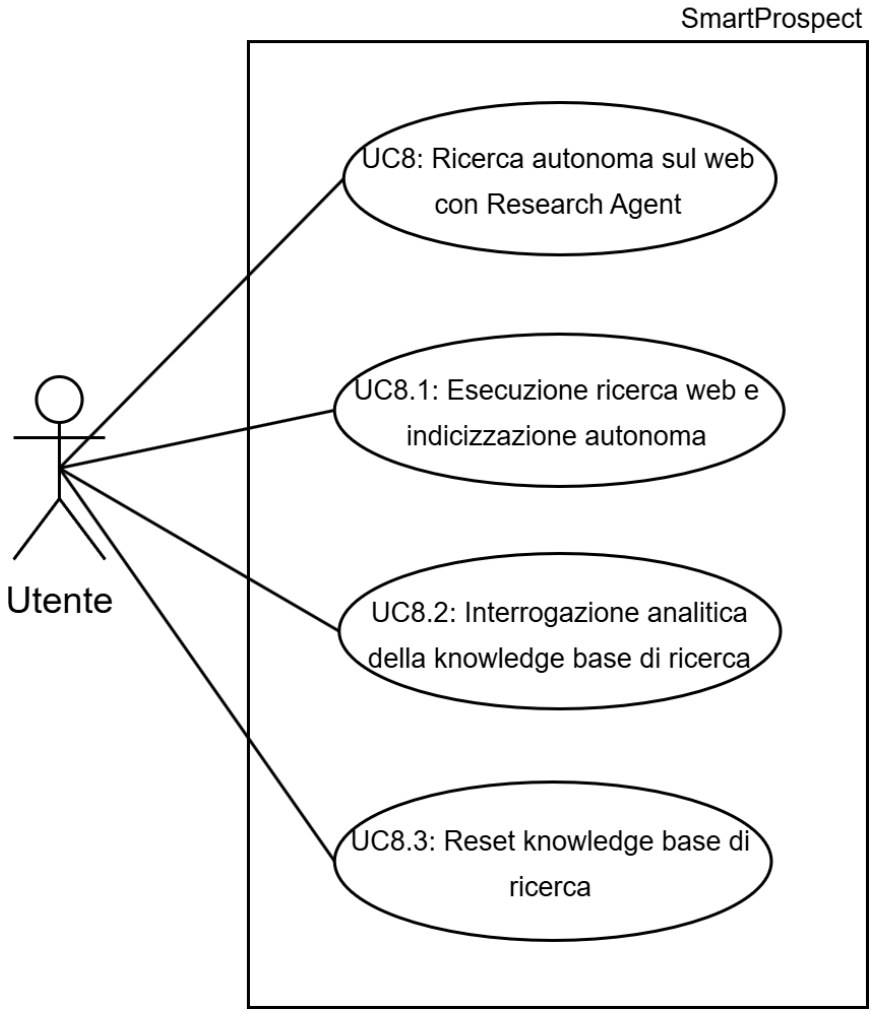

Figura 3.3: Diagramma del caso d'uso UC-8

UC8.1: Esecuzione ricerca *web* e indicizzazione autonoma

Attori Principali: Utente.
Precondizioni: L'utente si trova nella *dashboard* del Research Agent (UC-8).
Descrizione: L'utente formula una richiesta di ricerca in linguaggio naturale. L'agente genera e mostra una risposta analitica strutturata inserendo le citazioni alle fonti.
Postcondizioni: Le pagine *web* individuate sono indicizzate nel *vector store* e la risposta dell'agente viene persistita nella cronologia della sessione.
Scenario Alternativo: Un URL selezionato non è raggiungibile o restituisce un contenuto vuoto: la risorsa viene saltata e l'agente prosegue l'analisi con le successive fonti della coda senza interrompere la *pipeline*.

UC8.2: Interrogazione analitica della *knowledge base* di ricerca

Attori Principali: Utente.
Precondizioni: L'utente si trova nella *dashboard* del Research Agent (UC-8) e la sessione corrente contiene almeno un documento indicizzato (vedi UC-8.1).
Descrizione: L'utente pone una domanda di approfondimento o di sintesi mirata esclusivamente sui dati già collezionati. Il sistema recupera i *chunk* di testo pertinenti e genera la risposta finale senza effettuare nuove *query* o chiamate esterne verso la rete internet.
Postcondizioni: La risposta analitica basata sulla KB locale è visualizzata a schermo e persistita nel *database* del sistema.
Scenario Alternativo: Nessun frammento di testo memorizzato nel *vector store* risulta rilevante per rispondere al quesito: l'agente notifica all'utente l'impossibilità di estrarre una risposta certa dai documenti della sessione, evitando fenomeni di allucinazione.

UC8.3: *Reset knowledge base* di ricerca

Attori Principali: Utente.
Precondizioni: L'utente si trova nella *dashboard* del Research Agent (UC-8) e ha all'attivo una sessione con documenti o messaggi memorizzati.
Descrizione: L'utente richiede esplicitamente la cancellazione e l'azzeramento della sessione di lavoro tramite l'apposita azione di interfaccia. Al termine dell'operazione, la *dashboard* viene ricaricata.
Postcondizioni: La sessione corrente viene integralmente distrutta; la *dashboard* si ripresenta nello stato iniziale con la cronologia dei messaggi svuotata e il contatore dei documenti configurato a 0.
Scenario Alternativo: Si verifica un errore bloccante sul *database* relazionale locale: il sistema interrompe la procedura, notifica il fallimento del *reset* all'utente tramite un messaggio di avviso e mantiene intatti i dati della sessione.

3.2.2 Tracciamento dei requisiti

Ho derivato i requisiti da tre fonti distinte:

- **i casi d'uso** (§3.2.1): da ciascun caso d'uso ho estratto i requisiti funzionali corrispondenti, trasformando scenario principale, scenari alternativi e postcondizioni in enunciati verificabili. Uno scenario alternativo genera

tipicamente un requisito autonomo, perché descrive un comportamento che il sistema deve esibire e che va quindi verificato separatamente;

- **il piano di lavoro** concordato con il *tutor* aziendale, dal quale discendono i requisiti di qualità (documentazione, diagrammi UML, *feedback* visivo delle operazioni asincrone) e i requisiti di vincolo tecnologico (*stack* Laravel, Filament, Docker ecc.);
- **i colloqui di revisione** al termine di ogni *sprint* (§3.1.2), che hanno raffinato priorità e richieste man mano che il progetto avanzava.

Per la definizione di ciascun requisito ho fatto riferimento a tre proprietà fondamentali: l'**atomicità**, intesa come la caratteristica per cui ogni requisito descrive una singola funzionalità verificabile, la **tracciabilità**, che garantisce la possibilità di ricondurre ogni requisito alla relativa fonte di origine e la **verificabilità**, ovvero la capacità di formulare il requisito in modo tale da poter stabilire oggettivamente il suo effettivo soddisfacimento.

Classificazione e codifica

I requisiti vengono classificati secondo le seguenti categorie:

- **Requisiti Funzionali (F)**: descrivono le funzionalità che il sistema deve offrire all'utente.
- **Requisiti di Qualità (Q)**: descrivono le caratteristiche qualitative attese del sistema.
- **Requisiti di Vincolo (V)**: descrivono i vincoli tecnologici adottati nel progetto.

Ogni requisito è identificato da un codice univoco nella forma:

`R-[ID]-[Tipo]-[Priorità]`

dove:

- **ID**: numero progressivo del requisito;
- **Tipo**: F (Funzionale), Q (Qualità), V (Vincolo);
- **Priorità**: Ob (Obbligatorio), De (Desiderabile), Op (Opzionale).

Esempi rappresentativi

La Tabella 3.2 riporta un campione rappresentativo dell'elenco dei requisiti, scelto per illustrare le tre categorie, i tre livelli di priorità e il legame con le fonti: i requisiti funzionali riportati derivano dai due casi d'uso presentati in §3.2.1, e mostrano come da un singolo caso d'uso (UC-7.2) possano discendere più requisiti atomici distinti.

Codice	**Descrizione**	**Fonte**
R-34-F-Ob	Il sistema deve permettere la navigazione della gerarchia di cartelle Google Drive e la selezione di PDF da indicizzare.	UC-7.2
R-35-F-Ob	Il sistema deve indicizzare in modo asincrono i PDF selezionati in un OpenAI *vector store*, aggiornando lo stato della *knowledge base* al termine.	UC-7.2
R-37-F-De	Il sistema deve permettere il *reset* completo della *knowledge base* RAG, eliminando conversazioni e metadati associati all'utente.	UC-7.4
R-39-F-Ob	Il sistema non deve elaborare nuovamente risorse *web* già indicizzate nell'ambito della stessa sessione di ricerca.	UC-8.1
R-3-Q-De	L'interfaccia utente deve fornire *feedback* visivo sullo stato delle operazioni asincrone in corso.	Piano di lavoro
R-4-Q-Op	L'interfaccia utente deve essere ottimizzata per la fruizione su dispositivi mobili.	Piano di lavoro
R-4-V-Ob	L'infrastruttura applicativa deve essere containerizzata tramite Docker.	Piano di lavoro

Tabella 3.2: Campione rappresentativo dei requisiti individuati, scelto per coprire le tre categorie (F, Q, V) e i tre livelli di priorità (Ob, De, Op). Fonte: `elaborazione propria`

Censimento complessivo e tracciamento

Il processo di derivazione ha prodotto **57 requisiti** complessivi, la cui distribuzione per categoria e priorità è riassunta nella Tabella 3.3.

Categoria	Obbligatori	Desiderabili	Opzionali	Totale
Funzionali (F)	43	5	0	**48**
Qualità (Q)	2	1	1	**4**
Vincolo (V)	5	0	0	**5**
Totale	**50**	**6**	**1**	**57**

Tabella 3.3: Censimento complessivo dei requisiti per categoria e priorità. Fonte: `elaborazione propria`

A valle della derivazione ho costruito una tabella di tracciamento bidirezionale requisiti ↔ fonti, mantenuta aggiornata per l'intera durata dello *stage* e utilizzata come strumento di controllo di completezza: al termine dell'analisi, tutti i 48 requisiti funzionali risultano riconducibili ad almeno uno dei 36 identificatori UC, e ciascun caso d'uso origina almeno un requisito; i requisiti di qualità e di vincolo sono tracciati sul piano di lavoro.

3.3 Progettazione e implementazione

3.3.1 Architettura e *stack* tecnico del progetto

Oltre allo *stack* tecnologico aziendale descritto nel Capitolo 1 (§1.2), SmartProspect adotta un'architettura in cui il nucleo applicativo Laravel è affiancato da una serie di servizi ausiliari, ciascuno isolato nel proprio *container* Docker e responsabile di una funzionalità specifica. Non si tratta di microservizi in senso stretto, poiché l'applicazione condivide un'unica *codebase* e un unico *database*: i servizi aggiuntivi sono da intendersi come strumenti specializzati che Laravel chiama quando ne ha bisogno.

I servizi ausiliari integrati nel progetto sono:

- Un servizio Python costruito con FastAPI, dedicato alle operazioni più pesanti dal punto di vista computazionale: la valutazione automatica della qualità visiva dei siti *web* tramite intelligenza artificiale e le attività di *scraping* più articolate.
- Crawl4AI, uno strumento di navigazione automatica del *web* in grado di eseguire pagine con contenuto dinamico, che Laravel utilizza ogni volta che deve analizzare un sito in modo approfondito.
- SearXNG, un motore di ricerca *self-hosted* che aggrega i risultati di più motori contemporaneamente e viene usato per le ricerche OSINT e per l'individuazione automatica dei *competitor*.

- GOSOM, uno strumento dedicato esclusivamente alla raccolta strutturata di dati da Google Maps: categorie merceologiche, siti *web* aziendali, informazioni geografiche.

Le operazioni di raccolta dati richiedono spesso tempi lunghi e non possono bloccare l'interfaccia utente mentre lavorano. Per questo Laravel le affida a dei *job* gestiti da Laravel Queue con Redis, che li esegue in *background* in modo ordinato e controllato.

Il servizio Python utilizza Pyppeteer per controllare un *browser* Chromium in modo automatico mentre per la valutazione visiva dei siti si appoggia a ResNet18, una rete neurale pre-addestrata in grado di estrarre caratteristiche visive da uno *screenshot* e confrontarle per stimarne la qualità.

La Figura 3.4 illustra come queste componenti si organizzano all'interno dell'infrastruttura Docker.

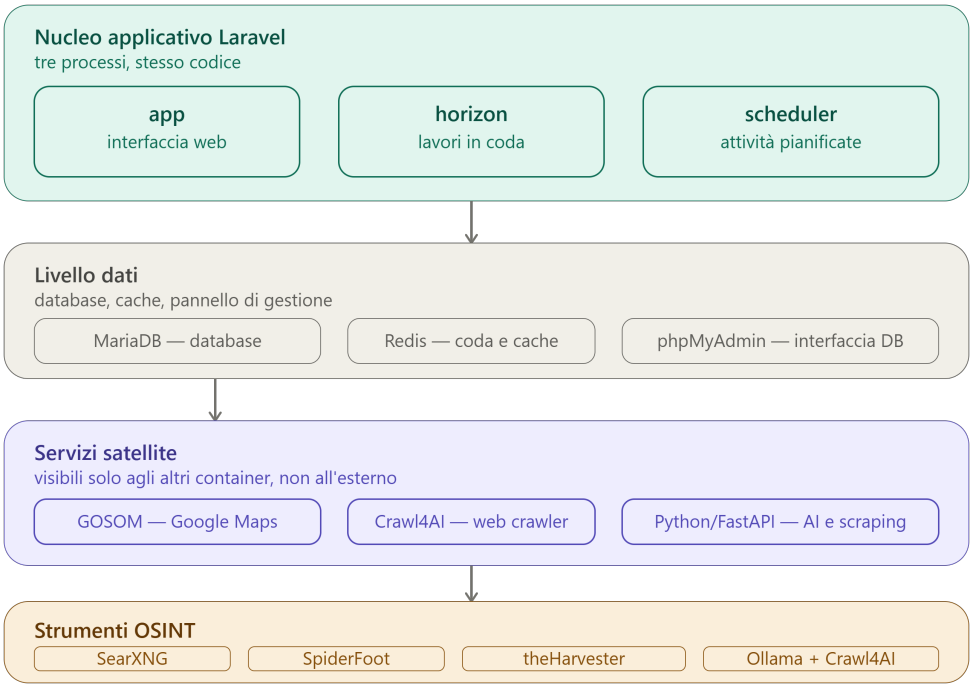

Figura 3.4: Organizzazione dei *container* Docker in ambiente di sviluppo: il nucleo applicativo Laravel, il livello dati, i servizi satellite accessibili solo internamente e gli strumenti OSINT. Fonte: `elaborazione propria`

3.3.2 Panoramica dei sottosistemi sviluppati

SmartProspect è articolato in numerosi sottosistemi di acquisizione e analisi dati. Tra questi, ne ho selezionati tre per un'analisi di dettaglio nelle sezioni successive (§3.3.3, §3.3.4, §3.3.5), in ragione della loro ricchezza architetturale

e delle conoscenze tecniche che ne hanno richiesto la realizzazione. I rimanenti, a minore complessità progettuale, li ho sintetizzati nella Tabella 3.4.

Sottosistema	**Scopo**	**Tecnologie chiave**
Scansione siti *web* e valutazione IA	Analizza il sito di un *competitor* (`ScanWebsiteJob`), acquisisce *screenshot* e classifica la qualità visiva tramite k-NN su *feature* ResNet18.	Spatie Crawler, Pyppeteer, PyTorch/ResNet18, Redis
Acquisizione dati da Google Maps	*Job* orchestratore che recupera categoria merceologica e sito *web* mancanti interrogando GOSOM con strategie *anti-ban*.	GOSOM, Laravel Queue
Intelligence Trend Amazon	*Crawler* Node.js periodico che traccia *rank*, prezzo e variazioni storiche tramite *snapshot* immutabili.	*Scraper* Node.js, Eloquent
Intelligence macroeconomica (GDP)	Importa indicatori dalla World Bank calcolando tassi di crescita annuali e aggregazioni per settore.	Artisan, API World Bank, Chart.js
Intelligence SEO e *competitor*	Monitora posizionamento organico, *backlink* e SEO *on-page*, generando *alert* automatici sui cali di prestazione.	Google Search Console, Crawl4AI, Open PageRank

Tabella 3.4: Panoramica dei sottosistemi a minore complessità progettuale. Fonte: `elaborazione propria`.

3.3.3 *Pipeline* OSINT di Company Lookup

`CompanyLookupOrchestratorJob` è il nucleo dell'indagine OSINT: riceve in ingresso uno dei cinque identificativi supportati (Partita IVA, dominio, email, ragione sociale o nome) e orchestra una catena di nove *job* sequenziali tramite `Bus::chain()` di Laravel. Ogni *job* si occupa di una sola responsabilità, passa i risultati al modello condiviso `CompanyLookupSearch` e lascia il controllo al successivo. L'avanzamento è visibile in tempo reale sull'interfaccia grazie a un meccanismo di *polling* Livewire ogni 3 secondi.

```php
Bus::chain([
    new FetchViesDataJob($this->search),
    new FindCompanyWebsiteJob($this->search),
    new CrawlCompanyWebsiteJob($this->search),
    new HarvestCompanyDomainJob($this->search),
    new WhoisCompanyDomainJob($this->search),
    new FetchFinancialDataJob($this->search),
    new CrawlCameraleSearchJob($this->search),
    new BreachCheckJob($this->search),
    new FinalizeCompanyLookupJob($this->search),
])->dispatch();
```

Codice 3.1: Definizione della catena sequenziale dei *job* tramite Bus Chain: ogni fase isola una responsabilità OSINT indipendente, illustrando in pratica l'applicazione del *pattern* Chain of Responsibility discusso in §3.3.6. Fonte: `elaborazione propria`

La catena si articola in quattro blocchi logici distinti.

Il **primo blocco** (fasi 1–5) raccoglie le informazioni pubbliche di base sull'azienda:

- **FetchViesDataJob** interroga il sistema VIES della Commissione Europea per verificare la validità fiscale della Partita IVA e recuperare denominazione ufficiale e sede legale.
- **FindCompanyWebsiteJob** cerca il sito aziendale tramite SearXNG, applica una *blacklist* di oltre 50 domini non pertinenti (*social, directory* testate) e assegna un punteggio di rilevanza basato su TLD e profondità dell'URL.
- **CrawlCompanyWebsiteJob** analizza l'*homepage* e fino a sei sotto-pagine tramite Crawl4AI, estraendo contatti, canali *social*, descrizioni e mappando il *tech stack* tramite oltre 30 firme tecnologiche.
- **HarvestCompanyDomainJob** e **WhoisCompanyDomainJob** completano l'analisi del dominio: il primo enumera sottodomini e indirizzi email tramite theHarvester (DuckDuckGo, Yahoo, crt.sh), il secondo recupera i dati storici di registrazione tramite le API WhoisXML.

Il **secondo blocco** (fase 6) aggrega i dati finanziari incrociando tre fonti: *lookup* diretto su OpenCorporates, *scraping* di *infoimprese.it* tramite Crawl4AI

per estrarre forma giuridica, capitale sociale, addetti, codice ATECO, fatturato, utile e patrimonio netto, e come *fallback* una ricerca mirata su una *whitelist* di 14 domini finanziari affidabili via SearXNG.

Il **terzo blocco** (fase 7) estrae i dati ufficiali del Registro delle Imprese da *ufficiocamerale.it*: il primo *job* individua la Partita IVA corretta scorrendo le sorgenti già popolate, il secondo effettua lo *scraping* della scheda aziendale con i filtri avanzati di Crawl4AI (`magic: true`), normalizza i formati numerici italiani e mappa la provincia alla regione tramite una *lookup table* interna di 107 province.

Il **quarto blocco** (fase 8–9) chiude la catena con `FinalizeCompanyLookupJob`, che calcola il Confidence Score: un indicatore numerico tra 0 e 100 che riflette completezza e affidabilità della ricerca. L'algoritmo è additivo e assegna pesi a ciascuna informazione raccolta:

Informazione	**Punti**
Validità VIES	+25
Sito individuato	+15
Email (bonus +3 se più di 2)	+10
PEC	+8
Utenze telefoniche (bonus +2 se più di 1)	+8
Dati finanziari strutturati (bonus +5 con fatturato)	+10
Dati da theHarvester	+7
Record WHOIS	+7
Codice Fiscale	+4
Indirizzi fisici, canali *social*, *stack* tecnologico, *markup* JSON-LD, TLD coerente	+5 ciascuno

Tabella 3.6: Pesi dell'algoritmo additivo del Confidence Score: a ciascuna informazione raccolta viene assegnato un punteggio parziale; il valore finale è normalizzato tramite `min(100, $totalPoints)` per restare sempre nel *range* $[0, 100]$. Fonte: `elaborazione propria`

La verifica della proprietà di monotonicità di questo algoritmo è descritta in §3.4.2.

3.3.4 Sistema RAG con Google Drive

Il modulo RAG permette di collegare il proprio Google Drive alla piattaforma, selezionare una cartella di documenti PDF e interrogarli in linguaggio naturale. Il flusso si articola in tre fasi: autenticazione con Google tramite

OAuth2, indicizzazione asincrona dei documenti in un *vector store* OpenAI, e *chat* in *streaming* con il modello LLM che risponde attingendo esclusivamente ai contenuti indicizzati.

Il *layer* IA è costruito sulla libreria Laravel AI (`Laravel\Ai`), che permette di definire agenti conversazionali tramite semplici interfacce PHP. La configurazione dell'agente avviene tramite attributi PHP dichiarati direttamente sulla classe:

```php
#[MaxSteps(5)]
#[MaxTokens(4096)]
#[Temperature(0.3)]
class DocumentRagAgent implements Agent, HasTools
{
    use Promptable;
    // ...
}
```

Codice 3.2: Dichiarazione dell'agente RAG tramite attributi PHP 8: il numero massimo di passi (5) e la temperatura bassa (0.3) riflettono un agente volutamente rigido, limitato ai soli documenti indicizzati. Fonte: `elaborazione propria`

Il modulo si articola in tre componenti principali. Il **primo componente** è `DocumentRagAgent`, l'agente conversazionale vero e proprio. Il suo *system prompt* impone tre vincoli espliciti: rispondere solo con informazioni presenti nei documenti, citare sempre il *file* sorgente, non inventare nulla. Il metodo `tools()` inietta il *tool* `FileSearch`, lo strumento che esegue la ricerca semantica sul *vector store*, ma solo se l'utente ha già sincronizzato almeno una cartella. Se la *knowledge base* non esiste ancora, la lista dei *tool* restituita è semplicemente vuota:

```php
public function tools(): iterable
{
    $kb = RagKnowledgeBase::where('user_id',
    ↪  $this->user->id)->first();
    if (! $kb) return [];

    return [
        new FileSearch(stores: [$kb->openai_vector_store_id]),
    ];
}
```

Codice 3.3: Iniezione condizionale del *tool* `FileSearch` in base alla presenza della *knowledge base*: se l'utente non ha ancora sincronizzato alcun documento, la lista dei *tool* restituita è vuota e l'agente non tenta alcuna ricerca. Fonte: `elaborazione propria`

Il **secondo componente** è `SyncFolderToVectorStore`, il *job* asincrono (*timeout* 600s, 2 tentativi) che gestisce l'intera *pipeline* di indicizzazione. Il flusso è semplice: se non esiste ancora un *vector store* per l'utente lo crea, poi per ogni PDF lo scarica da Drive, lo carica su OpenAI e lo collega al *vector store*. Una precauzione adottata è che ogni *file* temporaneo viene eliminato con `@unlink()` sia al termine corretto sia in caso di errore, così nessun *file* residuo rimane sul *server*:

```php
foreach ($this->files as $file) {
    $tmpPath = sys_get_temp_dir().'/'.$file['id'].'.pdf';
    try {
        $pdfBytes = $drive->downloadPdf($file['id']);
        file_put_contents($tmpPath, $pdfBytes);
        unset($pdfBytes);  // libera memoria subito

        $fileId = Http::withToken($apiKey)
            ->attach('file', file_get_contents($tmpPath),
            ↪  $file['name'])
            ->post('https://api.openai.com/v1/files', ['purpose' =>
            ↪  'assistants'])
            ->json('id');

        Http::withToken($apiKey)->withHeaders($headers)
            ->post("/v1/vector_stores/{$storeId}/files", ['file_id' =>
            ↪  $fileId]);

        @unlink($tmpPath);
        $uploaded++;
    } catch (\Throwable $e) {
        @unlink($tmpPath);  // pulizia garantita nel catch
    }
}
```

Codice 3.4: *Pipeline* di *upload* PDF con gestione garantita dei *file* temporanei: `@unlink()` viene chiamato sia al termine dell'*upload* riuscito sia nel blocco `catch`, assicurando che nessun *file* temporaneo rimanga sul *filesystem* in caso di errore. Fonte: `elaborazione propria`

Il **terzo componente** è l'interfaccia utente, composta da tre pagine Filament: `RagIndex` mostra lo stato della *knowledge base* (connessa o meno, numero di documenti indicizzati); `RagFolders` permette di navigare le cartelle Drive e avviare la sincronizzazione; `RagChat` espone l'interfaccia di conversazione. Quest'ultima usa il *pattern* Server-Sent Events (SSE) per mostrare la risposta progressivamente mentre viene generata: il messaggio dell'utente viene salvato nel *database* subito, prima ancora che lo *stream* inizi, mentre la risposta dell'agente viene salvata solo a *stream* concluso. Questo ordine garantisce che un eventuale errore di rete non lasci messaggi senza risposta nel *log*:

```php
public function stream(Request $request)
{
    $request->validate(['message' => 'required|string|max:2000']);

    RagConversation::create([
        'user_id' => $request->user()->id,
        'role'    => 'user',
        'content' => $request->input('message'),
    ]);

    return (new DocumentRagAgent($request->user()))
        ->stream($request->input('message'));
}
```

Codice 3.5: *Pattern* SSE con *split* salvataggio: il messaggio utente viene persistito nel *database* prima di avviare lo *stream*, evitando che un errore di *streaming* lasci messaggi orfani senza risposta nel *log* della conversazione. Fonte: `elaborazione propria`

3.3.5 Research Agent: ricerca autonoma sul *web*

Mentre il sistema RAG lavora su documenti caricati dall'utente, il `ResearchAgent` parte da zero: riceve una domanda, cerca autonomamente sul *web*, salva quello che trova e risponde basandosi sui contenuti raccolti. È configurato con `MaxSteps(15)` proprio perché il suo ciclo di lavoro è più lungo: cercare, leggere le pagine, salvare, cercare ancora con *query* correlate. La temperatura leggermente più alta (0.4 *vs* 0.3) lascia all'agente più margine per connettere informazioni provenienti da fonti diverse.

Il modulo si articola in quattro *tool* distinti.

Il **primo**, interno all'agente, genera il *system prompt* dinamicamente: il numero di documenti già indicizzati nella sessione viene iniettato nel testo, così l'agente sa sempre quante fonti ha già a disposizione prima di decidere se cercarne di nuove. Il *tool* `FileSearch` viene aggiunto alla lista solo se esiste già un *vector store* per la sessione corrente:

```php
public function instructions(): string
{
    $session = ResearchSession::forUser($this->user->id);
    $docsCount = $session?->total_documents ?? 0;

    return <<<PROMPT
    Sei un agente di ricerca intelligente. ...
    - file_search: cerca nella knowledge base già costruita
    ↪  ({$docsCount} documenti indicizzati)
    ...
    Knowledge base attuale: {$docsCount} documenti indicizzati.
    PROMPT;
}
```

Codice 3.6: Generazione dinamica del *system prompt*: il conteggio dei documenti già indicizzati viene iniettato nel testo, così l'agente conosce il contesto disponibile prima di ogni risposta. Fonte: `elaborazione propria`

Il **secondo** è `SearchWeb`, che interroga SearXNG aggregando quattro motori (`google`, `bing`, `duckduckgo`, `ahmia`) e restituisce all'agente un JSON con titolo, URL e *snippet* per ogni risultato. I risultati privi di URL vengono filtrati prima della restituzione:

```php
public function handle(Request $request): string
{
    $results = collect($response->json('results', []))
        ->take($request['maxResults'] ?? 5)
        ->map(fn ($r) => [
            'title'   => $r['title'] ?? 'N/D',
            'url'     => $r['url'] ?? '',
            'snippet' => $r['content'] ?? '',
        ])
        ->filter(fn ($r) => ! empty($r['url']))
        ->values()->toArray();

    return json_encode(['query' => $query, 'results' => $results,
    ↪  'count' => count($results)]);
}
```

Codice 3.7: Implementazione del *tool* `SearchWeb`: aggrega i risultati di quattro motori tramite SearXNG e filtra quelli privi di URL prima di restituire la lista all'agente. Fonte: `elaborazione propria`

Il **terzo** è `ScrapeUrl`, che legge il contenuto di una pagina tramite Crawl4AI e lo tronca a 8.000 caratteri, limite scelto per rispettare la finestra di contesto dell'LLM senza perdere le parti più rilevanti del testo.

Il **quarto**, `SaveResearch`, è il più complesso: prima di salvare un contenuto controlla che l'URL non sia già stato indicizzato in questa sessione, crea il *vector store* in modo *lazy* se non esiste ancora, carica il *file* su OpenAI e aggiorna il contatore:

```php
if (ResearchDocument::alreadyScraped($this->userId, $url)) {
    return json_encode(['status' => 'skipped', 'reason' => 'URL già
    ↪ indicizzata']);
}

$tmpPath = sys_get_temp_dir().'/research_'.md5($url).'.txt';
file_put_contents($tmpPath, "URL: $url\nTitolo: $title\n\n$content");

$fileId = Http::withToken($apiKey)
    ->attach('file', file_get_contents($tmpPath), basename($tmpPath))
    ->post('https://api.openai.com/v1/files', ['purpose' =>
    ↪ 'assistants'])
    ->json('id');

$session->increment('total_documents');
```

Codice 3.8: Logica di deduplicazione per URL, creazione *lazy* del *vector store* e *upload* del contenuto in `SaveResearch`: gli URL già indicizzati vengono saltati silenziosamente senza interrompere il flusso dell'agente. Fonte: `elaborazione propria`

L'interfaccia utente è la *dashboard* `ResearchChat`, che mostra il numero di documenti raccolti, la cronologia delle conversazioni e un pulsante di *reset* che cancella in sequenza il *vector store*, la sessione, i documenti e le conversazioni. Il controller segue lo stesso *pattern* SSE del sistema RAG, con l'unica differenza che istanzia `ResearchAgent` al posto di `DocumentRagAgent`.

3.3.6 *Design pattern* utilizzati

L'architettura del sistema SmartProspect fa un uso consapevole dei *design pattern* classici implementati sia direttamente sia attraverso le astrazioni fornite dal *framework* Laravel. Nella tabella sottostante analizzo i più rilevanti presenti nel codice:

Pattern	**Dove è usato**	**Problema che risolve**
Observer	`Spatie\Crawler`, Observer Eloquent	Reagire a eventi (*download* pagine, salvataggio modelli) senza accoppiare chi genera l'evento a chi lo gestisce.
Chain of Responsibility	`CompanyLookup` `OrchestratorJob`	Orchestrare una catena di operazioni sequenziali indipendenti, ciascuna con una sola responsabilità.
Strategy	`NewsAggregator`	Intercambiare sorgenti di *news* (NewsAPI, GNews) senza modificare il codice che le usa.
Facade	Laravel (`Http`, `Log`, `Cache`...)	Invocare componenti complesse tramite un'interfaccia semplice, mantenendo la possibilità di sostituirle nei *test*.
Singleton	Microservizio Python, Pyppeteer	Mantenere una sola istanza del *browser* Chromium attiva tra le richieste, evitando 2–3 secondi di avvio ad ogni chiamata.
Command	Comandi Artisan (`gdp:fetch-worldbank`, `seo:check-alerts`...)	Incapsulare operazioni di manutenzione in oggetti autonomi richiamabili da riga di comando o da *scheduler*.
Proxy	`PythonServiceClient`, `Crawl4AIService`, `SearXNGService`	Nascondere al resto del codice i dettagli di comunicazione HTTP con i *container* Docker remoti.
Template Method	`Promptable`, agenti IA	Definire un flusso fisso di orchestrazione condiviso, lasciando alle sottoclassi solo la specializzazione dei parametri.

Tabella 3.7: *Design pattern* applicati nel progetto: per ciascuno sono indicati il contesto d'uso e il problema concreto che risolve. Fonte: `elaborazione propria`.

3.4 Verifica e validazione

3.4.1 *Framework* e strategia di *test*

La strategia di *testing* si basa su **Pest PHP**, un *framework* costruito sopra PHPUnit con una sintassi più fluente e leggibile. Tutti i *test* si trovano nella

directory `tests/Feature/` e privilegiano i *test* di integrazione rispetto a quelli unitari puri: in un sistema come SmartProspect, dove la logica di *business* è strettamente legata allo stato del *database*, alle code e alle API esterne, testare i flussi completi è più significativo che testare singole funzioni in isolamento.

Per rendere i *test* affidabili e indipendenti dall'ambiente reale, vengono usati sistematicamente cinque meccanismi di supporto:

- **RefreshDatabase**: ogni *test* parte da un *database* pulito, grazie a una transazione con *rollback* automatico al termine.
- **Bus::fake()** e **Queue::fake()**: intercettano i *job* prima che vengano eseguiti, permettendo di verificare che vengano accodati con i parametri giusti senza dover aspettare che girino davvero.
- **Http::fake()**: sostituisce le chiamate HTTP verso API esterne con risposte predefinite, eliminando la dipendenza dalla rete durante i *test*.
- **Mockery**: sostituisce i *client* verso i servizi esterni (`Crawl4AIService`, `SearXNGService`) con versioni controllate che rispondono come vogliamo.
- **Livewire::test()**: simula le interazioni con le componenti Filament senza aprire un *browser* reale.

3.4.2 Verifica della *pipeline* OSINT di Company Lookup

La *suite* `CompanyLookupTest.php` è la più estesa del progetto, con oltre 50 casi che coprono l'intero ciclo di vita della *pipeline* descritta in §3.3.3: *factory* di stato del modello, rilevamento automatico del tipo di *input*, comportamento condizionale dei *job*, deduplicazione dei contatti e unicità delle fonti.

Il listato seguente mostra un caso rappresentativo: verifica che una ricerca completata abbia uno stato coerente e un Confidence Score nel *range* atteso.

```php
it('factory state completed imposta status e confidence_score',
↪  function () {
    $search = CompanyLookupSearch::factory()->completed()->create();

    expect($search->status)->toBe(CompanyLookupStatus::Completed)
        ->and($search->confidence_score)->toBeGreaterThanOrEqual(40)
        ->and($search->confidence_score)->toBeLessThanOrEqual(100);
});
```

Codice 3.9: Verifica che una ricerca completata abbia stato `Completed` e un Confidence Score compreso nel *range* $[40, 100]$. Fonte: `elaborazione propria`

L'algoritmo del Confidence Score (§3.3.3) viene validato tramite *test* comparativi che accertano la **proprietà di monotonicità**: aggiungere un dato corretto non deve mai abbassare il punteggio. Il *test* seguente ne è un esempio concreto: confronta due ricerche identiche, una con PEC e una senza, e verifica che quella con PEC ottenga un punteggio maggiore.

```php
it('confidence score con PEC è maggiore di quello senza PEC', function
↪  () {
    $searchNoPec = CompanyLookupSearch::factory()->create([
        'web_contacts' => ['pec' => []],
    ]);
    (new FinalizeCompanyLookupJob($searchNoPec))->handle();
    $searchNoPec->refresh();

    $searchWithPec = CompanyLookupSearch::factory()->create([
        'web_contacts' => ['pec' => ['test@pec.it']],
    ]);
    (new FinalizeCompanyLookupJob($searchWithPec))->handle();
    $searchWithPec->refresh();

    expect($searchWithPec->confidence_score)
        ->toBeGreaterThan($searchNoPec->confidence_score);
});
```

Codice 3.10: *Test* comparativo per la proprietà di monotonicità: la presenza della PEC deve incrementare il Confidence Score, non ridurlo. Fonte: `elaborazione propria`

Lo stesso approccio è applicato a tutti i fattori contributivi (validità VIES, dati finanziari, *record* WHOIS, ecc.), verificando che la combinazione completa dei dati saturi il punteggio a ≥ 70 e che il risultato rimanga sempre nel *range* $[0, 100]$. Un *test* dedicato verifica anche la precedenza delle fonti: i dati strutturati di *infoimprese.it* non devono essere sovrascritti dai frammenti meno affidabili degli *snippet* SERP.

3.4.3 Verifica dei sottosistemi a minore complessità

Per i sottosistemi descritti in §3.3.2 riporto una sintesi della strategia di *test*, senza mostrarne il codice.

- **Scansione siti *web***: `DispatchWebsiteScansJobTest` verifica che l'orchestratore accodi i *job* solo per i siti con al più una pagina salvata, e che il *job* di geolocalizzazione sia idempotente. I *test* Livewire coprono il salvataggio del voto manuale e l'avanzamento alla risorsa successiva.
- **Trend Amazon**: `AmazonTrendsCommandTest` inietta un *file* di *fixture* JSON statico nel comando Artisan e verifica il numero di *snapshot* generati e il valore nullo del *delta rank* alla prima acquisizione.
- ***Scraping* finanziario e *news***: `CompanyWebScraperServiceTest` isola tramite `Http::fake()` il parsing HTML di *ufficiocamerale.it* e *fatturatoitalia.it*; `FetchNewsCommandTest` verifica il comportamento del *flag* `-bypass-cache`.
- **Descrizioni aziendali IA**: `AgenticCompanyTest` verifica i filtri di inclusione e la tenuta a fronte di fallimenti del *bus*.

I sottosistemi di acquisizione dati da Google Maps, *intelligence* macroeconomica e *intelligence* SEO non dispongono al momento di *test* automatizzati dedicati.

3.4.4 Verifica del sistema RAG e del Research Agent

I due moduli IA descritti nelle sezioni precedenti condividono la stessa logica di *test*: si verifica il comportamento in isolamento, si sostituiscono i servizi esterni con risposte simulate e si controllano i casi limite senza dover avviare un modello reale. Di seguito riporto i casi più rappresentativi.

Isolamento per utente. I metodi statici di `RagConversation` vengono testati verificando che operino esclusivamente sui dati dell'utente corretto:

```
it('clearForUser non cancella le conversazioni di altri utenti',
↪  function () {
    $u1 = User::factory()->create();
    $u2 = User::factory()->create();
    RagConversation::create(['user_id' => $u1->id, 'role' => 'user',
    ↪  'content' => 'Ciao']);
    RagConversation::create(['user_id' => $u2->id, 'role' => 'user',
    ↪  'content' => 'Hello']);

    RagConversation::clearForUser($u1->id);
    expect(RagConversation::where('user_id',
    ↪  $u1->id)->count())->toBe(0)
        ->and(RagConversation::where('user_id',
        ↪  $u2->id)->count())->toBe(1);
});
```

Codice 3.11: Verifica che `clearForUser()` non tocchi le conversazioni di altri utenti. Fonte: `elaborazione propria`

***Mock* di dipendenze esterne.** Il *job* `SyncFolderToVectorStore` è testato sostituendo `GoogleDriveService` con un *mock* e intercettando le chiamate OpenAI con `Http::fake()`, verificando che lo stato transiti correttamente da `syncing` a `ready` e che l'ID del *vector store* venga persistito. Lo stesso approccio è usato per `SearchWeb` (risposta SearXNG controllata, verifica struttura JSON e filtro URL vuoti) e per `ScrapeUrl` (troncamento a 8 000 caratteri e gestione del *markdown* sia come stringa che come *array* di *chunk*).

***Tool* condizionali.** Sia `DocumentRagAgent` sia `ResearchAgent` espongono un numero variabile di *tool* in base allo stato della sessione. Il *test* verifica i casi limite senza invocare l'LLM:

```php
it('ResearchAgent aggiunge FileSearch quando esiste un vector store',
↪  function () {
    $user = User::factory()->create();
    ResearchSession::create([
        'user_id'                => $user->id,
        'openai_vector_store_id' => 'vs_research_123',
        'total_documents'        => 5,
    ]);
    $tools = iterator_to_array((new ResearchAgent($user))->tools());
    expect($tools)->toHaveCount(4)

        ↪  ->and(collect($tools)->last())->toBeInstanceOf(FileSearch::class);
});
```

Codice 3.12: Con *vector store* attivo l'agente espone 4 *tool*; senza sessione ne espone 3, senza invocare l'LLM. Fonte: `elaborazione propria`

Deduplicazione URL e *reset*. `alreadyScraped()` verifica che URL già indicizzati vengano rilevati senza falsi positivi. `resetKnowledgeBase()` cancella in cascata sessione, documenti e conversazioni; la chiamata OpenAI per la rimozione del *vector store* è *best-effort*: eventuali errori vengono ignorati senza bloccare il *reset* locale.

3.4.5 Copertura e metodologia di validazione

La strategia di *test* si articola su cinque livelli, ciascuno pensato per coprire una categoria diversa di rischio:

- **Integrazione (Pest/PHPUnit)**: copre la logica di: *pipeline* OSINT, algoritmi di *scoring*, smistamento dei *job*, il tutto su un *database* SQLite *in-memory* per velocizzare l'esecuzione.
- **Componenti UI (Livewire::test)**: i moduli Filament sono testati simulando interazioni reali senza aprire un *browser*.
- **Chiamate HTTP (Http::fake)**: le integrazioni verso fonti esterne usano risposte simulate, rendendo i *test* indipendenti dalla rete e riproducibili in ambienti di *continuous integration*.
- **Regressione (Fixture JSON)**: il formato dei dati scambiati tra il *backend* PHP e lo *scraper* Node.js per i mercati Amazon è definito in un *file*

JSON fisso. Se lo *scraper* viene modificato e il suo *output* cambia struttura, il *test* lo rileva immediatamente confrontando il risultato con il *file* di riferimento.

- **Invarianti di *scoring***: il Confidence Score è validato come proprietà relativa invece che come valore assoluto, così i *test* restano validi anche se i coefficienti vengono ricalibrati in futuro.

Dati quantitativi di copertura

Per dare una misura oggettiva dell'estensione dell'attività di verifica, la Tabella 3.8 riassume i dati della *suite* automatizzata, misurati al termine dello *stage* eseguendo l'intera *suite* con Pest su *database* SQLite *in-memory*.

Modulo verificato	***File* di *test***	**Casi di *test***	**Copertura di linea**
Pipeline OSINT Company Lookup	3	∼83	∼35 %
Ricerca *dark web* (UC-3.3)	2	9	∼50 %
Sistema RAG (modello conversazione)	1	3	∼12 %
Research Agent (*tool*, sessioni)	1	3	∼30 %
Totale *suite*	**7**	**∼98**	**∼33 %**

Tabella 3.8: Dati quantitativi della *suite* di *test* automatizzata: numero di *file* e di casi di *test* per modulo e copertura di linea. Fonte: `elaborazione propria`

L'intera *suite* viene eseguita in circa 15 secondi su *database* SQLite *in-memory*, con circa 270 *assertion* complessive. I valori di copertura risultano contenuti per una ragione strutturale: gran parte del codice gestisce casi limite legati a risposte anomale di API esterne (*timeout*, *payload* malformati, autenticazioni scadute), percorsi che nei *test* vengono simulati solo parzialmente tramite `Http::fake()` e *mock*; la copertura delle classi di orchestrazione risente inoltre del fatto che i *job* della *pipeline* vengono testati singolarmente, lasciando fuori i percorsi attivati solo dalla catena completa in produzione. La copertura riportata si riferisce alle classi sviluppate durante lo *stage*: i sottosistemi di acquisizione dati da Google Maps, *intelligence* macroeconomica e *intelligence* SEO, come indicato in §3.4.3, non dispongono di *test* automatizzati dedicati e sono stati validati manualmente tramite sessioni di collaudo esplorativo sull'ambiente di sviluppo.

3.5 Risultati raggiunti

3.5.1 Panoramica qualitativa del prodotto

Come già detto, lo *stage* svolto presso Spazio Dev ha avuto come oggetto lo sviluppo di SmartProspect, una piattaforma il cui scopo è supportare processi di *business intelligence*. Nel corso di queste otto settimane ho realizzato una *pipeline* completa che copre l'intero ciclo di vita del dato: acquisizione automatizzata da fonti eterogenee, elaborazione asincrona in *background*, persistenza strutturata e consultazione tramite un'interfaccia amministrativa.

Dal punto di vista dell'utente, SmartProspect si presenta come un unico ambiente di lavoro in cui è possibile raccogliere, incrociare e interrogare informazioni su aziende e mercati senza dover accedere manualmente a fonti diverse. Le operazioni che richiedono tempi lunghi, come scansioni, *scraping* e sincronizzazioni con servizi esterni, avvengono in *background* senza bloccare l'interfaccia, così l'utente può avviare una ricerca e continuare a lavorare mentre il sistema raccoglie i dati.

Il prodotto finale è una piattaforma che trasforma un processo altrimenti frammentato e manuale, ad esempio consultare registri pubblici o cercare informazioni sui *competitor*, in un flusso automatizzato e consultabile da un'unica interfaccia, con un indicatore esplicito di affidabilità dei dati raccolti e la possibilità di interrogare i contenuti in linguaggio naturale tramite i due moduli IA.

3.5.2 Risultati quantitativi

Copertura dei requisiti. Il censimento di §3.2.2 documenta 57 requisiti complessivi, 48 funzionali, 4 di qualità e 5 di vincolo, riconducibili agli 11 casi d'uso principali censiti in §3.2.1, ciascuno articolato in varianti specifiche per un totale di 36 identificatori UC. La copertura bidirezionale requisiti $\leftrightarrow$ casi d'uso è completa: nessun requisito è rimasto privo di un caso d'uso che lo motivi, e nessun caso d'uso è privo di almeno un requisito che lo realizzi.

Codice prodotto. Il contributo sviluppato durante lo *stage* ammonta a circa 50 nuove classi PHP, distribuite tra *job* di orchestrazione (11 per la sola *pipeline* OSINT di Company Lookup, ai quali si aggiungono quelli dei moduli Amazon Intelligence, SEO Intelligence e agenti IA), modelli Eloquent (circa 12, tra cui i modelli di sessione e conversazione dei due agenti IA e il modello `CompanyLookupSearch`), pagine e *controller* Filament (10), servizi di integrazione verso i *container* Docker (4) e agenti IA (2). La stima in righe di codice

effettivo prodotto *ex novo* è di circa 10.000 righe complessive tra classi PHP, viste Blade e migrazioni, su un *repository* che al termine del tirocinio conta circa 800 *file* PHP per circa 80.000 righe, a testimonianza dell'ampiezza del progetto preesistente in cui il lavoro si è inserito.

***Test* automatizzati.** Il modulo Company Lookup dispone della *suite* di *test* più estesa del progetto: 3 *file* per un totale di circa 83 casi automatizzati, che coprono la logica del modello `CompanyLookupSearch` (rilevamento automatico del tipo di *input*, calcolo del Confidence Score, gestione dei dati finanziari, OSINT e violazioni), il comportamento condizionale dei *job* della *pipeline* OSINT e i servizi di *geo-lookup*, con integrazioni simulate tramite `Http::fake()` e *mock*.

Documentazione prodotta. Oltre al codice, ho redatto due documenti di natura tecnica: un manuale utente che descrive le funzionalità della piattaforma, il quale potrà essere consultato dagli operatori che in azienda sfrutteranno SmartProspect, e una specifica tecnica che potrà essere utilizzata come riferimento per la manutenzione e lo sviluppo futuro.

Casi d'uso realizzati. Gli 11 casi d'uso principali coprono l'intero spettro funzionale della piattaforma: autenticazione (UC-0), gestione anagrafica e progetti (UC-1, UC-2), modulo *intelligence* (UC-3), gestione vendite (UC-4), analisi *competitor* (UC-5), questionari (UC-6) e i quattro moduli introdotti durante lo *stage*, *knowledge base* RAG (UC-7), Research Agent (UC-8), Amazon Intelligence (UC-9) e SEO Intelligence (UC-10).

Nel prossimo Capitolo valuto in che misura ho effettivamente raggiunto gli obiettivi prefissati all'inizio del tirocinio, tanto da Spazio Dev quanto sul piano della mia crescita personale, attraverso una valutazione oggettiva formulata a conclusione di questa tesi.

Capitolo 4

Valutazione retrospettiva

Con questo Capitolo concludo la relazione, offrendo una visione sinottica dell'esperienza di tirocinio. Analizzo la realizzazione degli obiettivi concordati nel Capitolo 2, la crescita professionale maturata durante il percorso e propongo alcune riflessioni sul modo in cui le conoscenze acquisite durante gli studi trovano applicazione all'interno di un contesto lavorativo.

4.1 Bilancio dell'esperienza formativa

Globalmente, ho raggiunto gli obiettivi definiti durante la pianificazione (Capitolo 2, §2.3). Grazie alla formazione iniziale ho acquisito dimestichezza con lo *stack* tecnologico e l'autonomia necessaria per contribuire allo sviluppo, mentre la struttura modulare del sistema, basata sulla separazione tra acquisizione, elaborazione e consultazione dei dati, mi ha consentito di integrare progressivamente nuove funzionalità senza compromettere la stabilità delle componenti già esistenti.

Le strategie di mitigazione individuate nell'analisi preventiva dei rischi (Capitolo 2, §2.4) si sono rivelate efficaci.

Gli obiettivi tecnici concordati con l'azienda nel piano di lavoro si articolavano in cinque aree di intervento. Al termine dello *stage*, il loro stato è il seguente:

1. **Sviluppo dei *job* di orchestrazione**: obiettivo raggiunto. Gli 11 *job* della *pipeline* OSINT di *Company Lookup* risultano compatibili con Laravel Horizon e coprono sia le sorgenti dati previste inizialmente sia fonti aggiuntive integrate durante lo sviluppo. Il corretto comportamento dei *job* è inoltre verificato tramite 83 casi di *test* automatizzati, che ne coprono la logica condizionale.

2. **Persistenza dei dati raccolti**: obiettivo raggiunto. La *pipeline* di trasferimento dai *container* di *scraping* al *database* MariaDB è operativa e i dati raccolti sono strutturati tramite modelli Eloquent con relazioni e migrazioni versionate. La copertura tra i 57 requisiti censiti e gli 11 casi d'uso principali risulta completa, senza requisiti relativi alla gestione dei dati privi di una corrispondente verifica.

3. **Normalizzazione e pulizia dei dati**: obiettivo raggiunto. I processi implementati consentono la deduplicazione dei contatti e la gestione delle anomalie nei *payload* delle API esterne tramite logiche di *fallback* a cascata. Il rischio relativo alla qualità e coerenza dei dati acquisiti, identificato nel Capitolo 2, si è verificato: le strategie introdotte per mitigarlo si sono dimostrate necessarie e sono rimaste parte integrante della *pipeline*.

4. **Interfaccia *web* di visualizzazione**: obiettivo raggiunto. Le 10 pagine e i *controller* Filament sviluppati coprono l'intero insieme degli 11 casi d'uso principali, includendo anche le viste relative ai moduli IA non previsti nel piano originale. Sono stati inoltre introdotti cruscotti interattivi per migliorare la consultazione dei dati elaborati.

5. ***Report* finale e presentazione al *team***: obiettivo raggiunto. La documentazione prodotta comprende un manuale utente e una specifica tecnica. Ho presentato i risultati ottenuti al *team* negli incontri conclusivi e il confronto con il *tutor* ha confermato la coerenza tra gli obiettivi inizialmente concordati e il lavoro effettivamente consegnato.

Guardando ora al percorso svolto, posso affermare che il tirocinio ha risposto positivamente alle aspettative che avevo maturato prima del suo avvio.

Uno degli aspetti che ritenevo più importanti era la possibilità di confrontarmi con un ecosistema tecnologico ampio e strutturato. Il lavoro svolto ha coinvolto diversi livelli dello *stack* Laravel, spaziando dalla gestione della *pipeline* OSINT alla realizzazione di moduli basati su intelligenza artificiale e interfacce di consultazione. La varietà delle attività affrontate ha quindi rispecchiato l'obiettivo iniziale di acquisire una visione complessiva del *framework*, senza limitarmi a un singolo ambito applicativo.

Un secondo elemento che desideravo approfondire riguardava la capacità di adattarmi a problematiche eterogenee. Nel corso del tirocinio ho affrontato attività differenti, che hanno richiesto approcci e competenze diverse: dall'integrazione con API esterne alla gestione della persistenza dei dati, fino alla realizzazione di interfacce e alla documentazione tecnica. Questa varietà si è rive-

lata un'importante occasione per sviluppare maggiore flessibilità nell'affrontare problemi non omogenei.

Particolarmente significativa è stata inoltre l'esperienza di operare all'interno di un progetto già esistente. Se durante il percorso universitario gran parte delle attività prevedeva la realizzazione di applicazioni progettate autonomamente, il tirocinio mi ha posto nella condizione di comprendere prima di tutto scelte architetturali, convenzioni e logiche definite da altri sviluppatori. L'integrazione del lavoro prodotto all'interno di SmartProspect mi ha mostrato quanto la capacità di leggere, interpretare ed estendere codice esistente sia una competenza centrale nello sviluppo professionale.

Anche sotto il profilo organizzativo l'esperienza ha avuto un impatto rilevante. Il confronto frequente con il *tutor* e la pianificazione progressiva delle attività mi hanno permesso di sperimentare un metodo di lavoro più organizzato rispetto a quello a cui ero abituata, imparando a gestire priorità, revisioni e momenti di verifica in funzione dell'avanzamento effettivo. L'andamento settimanale documentato in §3.1.1 evidenzia un progresso costante, con attività completate in anticipo in più occasioni e un solo rallentamento successivamente recuperato.

Infine, il tirocinio mi ha permesso di osservare direttamente le dinamiche dello sviluppo *software* in ambito aziendale. L'esperienza ha confermato il mio interesse verso questo settore, offrendo al tempo stesso una prospettiva più consapevole sulle responsabilità, sulle metodologie e sulle competenze necessarie per operare in un contesto professionale.

4.2 Competenze e conoscenze a confronto

L'esperienza di *stage* si è rivelata molto utile e ha rappresentato un'importante occasione di confronto con le dinamiche proprie dello sviluppo *software* in ambito professionale. Il contesto aziendale mi ha permesso di contribuire a un progetto con un impatto concreto sul *business*, affrontando problematiche reali e toccando con mano la complessità di un sistema come SmartProspect. Il percorso svolto ha inoltre rafforzato la mia consapevolezza del ruolo fondamentale di una solida strategia di *testing* all'interno del ciclo di sviluppo, soprattutto nella gestione di applicazioni articolate. Infine, l'esperienza mi ha avvicinata al mondo delle tecnologie di intelligenza artificiale, suscitando interesse verso un ambito che in precedenza non avevo avuto modo di approfondire.

L'università mi ha fornito solide fondamenta teoriche sulle quali costruire le competenze necessarie per affrontare le sfide del mondo *software*. Al tempo

stesso, è inevitabile che un percorso accademico non possa abbracciare nella sua interezza un ambito vasto e in continua evoluzione come quello dell'informatica. Durante gli anni di studio ho avuto l'opportunità di approfondire molteplici aspetti della disciplina: dall'analisi teorica dei componenti *hardware* e *software*, alla programmazione a basso livello, fino alla progettazione e allo sviluppo di applicazioni complete.

Ciò che, a mio avviso, è mancato è stato un elemento capace di collegare queste conoscenze in una visione unitaria. L'approccio necessariamente suddiviso in insegnamenti distinti mi ha consentito di acquisire competenze approfondite nei singoli ambiti, ma mi ha offerto solo in misura limitata l'occasione di comprenderne le interconnessioni all'interno di un contesto reale e complesso. Per questo motivo, attività maggiormente orientate alla pratica, come laboratori interdisciplinari o *workshop* progettuali, avrebbero potuto favorire la costruzione di una prospettiva più organica e integrata.

In questo senso, l'esperienza di tirocinio ha rappresentato il naturale completamento del percorso formativo. Essa ha svolto il ruolo di collegamento tra teoria e pratica, permettendomi di trasformare conoscenze fino ad allora prevalentemente astratte in strumenti concreti per la risoluzione di problemi reali. Mi ha insegnato l'importanza del metodo, della collaborazione e dell'adattabilità, mostrando come competenze apparentemente separate possano convergere verso un obiettivo comune.

Soprattutto, mi ha permesso di comprendere l'ampiezza delle opportunità offerte da questo settore e la profondità delle conoscenze che esso racchiude. Se da un lato questa esperienza ha rafforzato la fiducia nelle competenze acquisite durante il percorso universitario, dall'altro mi ha resa ancora più consapevole di quanto sconfinato sia il panorama dell'informatica e di quante prospettive rimangano da esplorare. Lungi dal rappresentare un punto di arrivo, tale consapevolezza costituisce per me uno stimolo a continuare ad approfondire le mie conoscenze e ad affrontare con spirito critico le sfide future, nella convinzione che la crescita professionale e personale passi attraverso una costante ricerca di nuove opportunità di apprendimento.

Ringraziamenti

Desidero esprimere la mia più sincera gratitudine al Professor **Tullio Vardanega**, relatore di questa tesi, per avermi guidata lungo questo percorso tanto delicato quanto significativo.

Un grazie profondo a **Mamma e Papà**, per aver sempre creduto in me, per il sostegno incondizionato e costante e per l'amore immenso con cui mi hanno accompagnata in ogni passo della mia vita. Senza di voi, nulla di tutto questo sarebbe stato possibile.

Grazie a mio fratello **Alex**, presenza insostituibile, che non mi ha mai lasciata sola, nella gioia e nel dolore, e che amo infinitamente.

Grazie a **Elisa**, compagna di crescita e di vita, con cui ho condiviso tanti ricordi e sogni e la cui amicizia sincera ha saputo resistere al tempo e ai cambiamenti.

Grazie a **Pierluigi**, che è stato un'ancora nei momenti più difficili, capace di offrirmi consigli, sostegno e leggerezza quando ne avevo più bisogno.

Grazie a **Gaia**, per la profonda amicizia che ci lega e per la meravigliosa persona che è, fonte costante di ispirazione e affetto.

Grazie a **Gabriel**, per il supporto fondamentale nel mio percorso accademico, per la cura e la disponibilità con cui mi ha affiancata e per la bellissima amicizia che è nata.

Grazie a **Daciana**, in cui ho scoperto una vera amica, che ha pianto con me nei miei giorni più tristi e ha riso con me nei miei giorni più felici.

Grazie a **Tutti i miei amici**, quelli di sempre e quelli incontrati lungo il cammino, che sono grata di avere nella mia vita. Ognuno di voi ha contribuito in modo unico a rendermi la persona che sono oggi; è anche grazie a voi se sono la Diana che conoscete.

Infine, un grazie a **me stessa**, per non essermi mai arresa. Ci sono stati giorni in cui ho toccato il fondo, ma dentro di me ho sempre trovato la forza di

risalire, anche quando la luce sembrava troppo lontana per essere vista. Sono orgogliosa di chi sono diventata e di dove sono arrivata.

AD MAIORA

Padova, Luglio 2026 *Diana Georgescu*

Bibliografia

Testi

Beck, Kent. *Test-Driven Development: By Example*. Addison-Wesley, 2002.

Fowler, Martin. *Patterns of Enterprise Application Architecture*. Addison-Wesley, 2002.

Gamma, Erich et al. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional, 1994.

Articoli

Cover, Thomas e Peter Hart. «Nearest Neighbor Pattern Classification». In: *IEEE Transactions on Information Theory* 13.1 (1967), pp. 21–27. DOI: 10.1109/TIT.1967.1053964.

He, Kaiming et al. «Deep Residual Learning for Image Recognition». In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2016), pp. 770–778. DOI: 10.1109/CVPR.2016.90.

Lewis, Patrick et al. «Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks». In: *Advances in Neural Information Processing Systems (NeurIPS)* 33 (2020). arXiv:2005.11401, pp. 9459–9474.

Pastor-Galindo, Javier et al. «The Not Yet Exploited Goldmine of OSINT: Opportunities, Open Challenges and Future Trends». In: *IEEE Access* 8 (2020), pp. 10282–10304. DOI: 10.1109/ACCESS.2020.2965257.

Sitografia

Alpine.js. *Alpine.js Documentation*. 2026. URL: https://alpinejs.dev/start-here (visitato il giorno 08/07/2026).

Commissione Europea. *VIES – VAT Information Exchange System*. 2026. URL: https://ec.europa.eu/taxation_customs/vies/ (visitato il giorno 10/06/2026).

Crawl4AI. *Crawl4AI Documentation*. 2026. URL: https://docs.crawl4ai.com/ (visitato il giorno 27/05/2026).

Docker Inc. *Docker Documentation*. 2026. URL: https://docs.docker.com/ (visitato il giorno 22/06/2026).

FastAPI. *FastAPI Documentation*. 2026. URL: https://fastapi.tiangolo.com/ (visitato il giorno 22/06/2026).

Filament. *Filament Documentation*. 2026. URL: https://filamentphp.com/docs (visitato il giorno 08/07/2026).

Gitea. *Gitea Documentation*. 2026. URL: https://docs.gitea.com/ (visitato il giorno 04/05/2026).

GNews. *GNews API v4 – Documentation*. 2026. URL: https://gnews.io/docs/v4 (visitato il giorno 07/05/2026).

Google. *Search Console API – Documentazione per gli sviluppatori*. 2026. URL: https://developers.google.com/webmaster-tools/search-console-api-original (visitato il giorno 03/06/2026).

Laracasts. *Laracasts – Laravel, PHP, and Vue.js video tutorials*. 2026. URL: https://laracasts.com/ (visitato il giorno 08/07/2026).

laramies. *theHarvester – OSINT Reconnaissance Tool*. 2026. URL: https://github.com/laramies/theHarvester (visitato il giorno 27/05/2026).

Laravel LLC. *Laravel 11.x Documentation*. 2026. URL: https://laravel.com/docs/11.x (visitato il giorno 08/07/2026).

— *Laravel Horizon – Official Documentation*. 2026. URL: https://laravel.com/docs/11.x/horizon (visitato il giorno 08/07/2026).

— *Laravel Nova – Official Documentation*. 2026. URL: https://nova.laravel.com/docs (visitato il giorno 08/07/2026).

Laravel LLC. *Livewire Documentation*. 2026. URL: https://livewire.laravel.com/docs (visitato il giorno 08/07/2026).

MariaDB Foundation. *MariaDB Server Documentation*. 2026. URL: https://mariadb.com/kb/en/documentation/ (visitato il giorno 22/06/2026).

NewsAPI. *NewsAPI – Documentation*. 2026. URL: https://newsapi.org/docs (visitato il giorno 07/05/2026).

OpenAI. *Assistants API – Overview*. 2026. URL: https://platform.openai.com/docs/assistants/overview (visitato il giorno 22/06/2026).

— *Vector Stores – API Reference*. 2026. URL: https://platform.openai.com/docs/api-reference/vector-stores (visitato il giorno 22/06/2026).

Pest. *Pest – An Elegant PHP Testing Framework*. 2026. URL: https://pestphp.com/docs/installation (visitato il giorno 08/07/2026).

pyppeteer. *Pyppeteer – Headless Chrome/Chromium Automation Library for Python*. 2026. URL: https://github.com/pyppeteer/pyppeteer (visitato il giorno 22/06/2026).

PyTorch. *torchvision.models – ResNet*. 2026. URL: https://pytorch.org/vision/stable/models/resnet.html (visitato il giorno 22/06/2026).

SearXNG. *SearXNG Documentation*. 2026. URL: https://docs.searxng.org/ (visitato il giorno 27/05/2026).

Spatie. *spatie/crawler – Crawl Webpages Following All Links*. 2026. URL: https://github.com/spatie/crawler (visitato il giorno 27/05/2026).

Symfony. *Symfony Documentation*. 2026. URL: https://symfony.com/doc/current/index.html (visitato il giorno 08/07/2026).

Tailwind Labs. *Tailwind CSS Documentation*. 2026. URL: https://tailwindcss.com/docs (visitato il giorno 08/07/2026).