

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



**Analisi comparativa di strumenti ETL e
sviluppo di una soluzione per l'integrazione di
dati**

Tesi di Laurea

Relatore

Prof. Zanella Marco

Laureando

Marcuzzo Luca
Matricola 2113198

Ringraziamenti

Desidero ringraziare il professor Zanella Marco, mio relatore, per la disponibilità e l'aiuto ricevuti durante la stesura dell'elaborato.

Ringrazio con affetto tutta la mia famiglia, in particolare i miei genitori, che mi hanno sempre sostenuto e non mi hanno mai fatto mancare nulla, e Irene e Simone, che mi sono stati vicini durante questo percorso.

Un ringraziamento speciale va a Giada, che durante questi anni mi ha sempre sostenuto con pazienza e in modo incondizionato.

Ringrazio inoltre Alessandro e Dorde per aver reso più piacevoli e divertenti le giornate universitarie, e il gruppo NullPointerException, con cui ho affrontato e superato il corso di Ingegneria del Software.

Ringrazio anche Sinesy per avermi dato l'opportunità di svolgere lo stage e tutto il suo team per avermi accolto calorosamente, in particolare il mio tutor Luca Biasotto, Alessandro Casetta e Michele Bozzo. Ringrazio poi Lisa Casagrande, con cui ho condiviso e portato a termine il progetto di stage.

Desidero infine ringraziare tutti gli altri amici che, anche se non nominati singolarmente, mi sono stati vicini e hanno condiviso con me tanti momenti durante questi anni.

Padova, Settembre 2026

Marcuzzo Luca

Sommario

Il presente documento descrive il lavoro svolto da Luca Marcuzzo durante uno stage di 304 ore presso l'azienda Sinesy S.r.l. di Oderzo (TV), a conclusione del percorso di laurea in Informatica.

Il progetto nasce dall'esigenza aziendale di migliorare la gestione dei processi di integrazione dati, in particolare quelli legati all'importazione di file forniti dai clienti e destinati al database aziendale. In questo contesto, l'obiettivo principale dello stage è stato analizzare, confrontare e sperimentare strumenti per la realizzazione di flussi ETL/ELT in grado di acquisire file CSV con strutture differenti, validarli, trasformarli e prepararli al caricamento nel database aziendale, con l'ulteriore obiettivo di implementare concretamente la soluzione ritenuta più adatta sulla base dell'analisi svolta.

L'importanza del progetto deriva dalla necessità di rendere più controllato, tracciabile, affidabile e automatizzato un processo che, se gestito manualmente o con strumenti poco integrati, può risultare complesso, ripetitivo e soggetto a errori. Un sistema di integrazione efficace permette infatti di ridurre il rischio di inserire dati incoerenti nel database, migliorare il monitoraggio delle elaborazioni, semplificare l'individuazione degli errori e supportare la gestione di file di grandi dimensioni provenienti da clienti diversi.

Indice

1	Introduzione	1
1.1	L'azienda	1
1.2	Il progetto	1
1.3	Organizzazione del testo	2
2	Descrizione dello stage	4
2.1	Contesto aziendale e organizzativo	4
2.2	Pianificazione delle attività	4
2.3	Metodo di lavoro adottato	5
2.4	Analisi dei rischi	5
3	Analisi dei requisiti	7
3.1	Problema da risolvere e obiettivi	7
3.2	Attori	8
3.3	Casi d'uso	9
3.3.1	UC01 – Autenticazione	9
3.3.2	UC02 – Visualizzazione errore credenziali non valide . . .	11
3.3.3	UC03 – Caricamento del file CSV	11
3.3.4	UC04 – Visualizzazione errore di caricamento del file . .	13
3.3.5	UC05 – Mappatura dei campi	14
3.3.6	UC06 – Visualizzazione errore colonna duplicata	17
3.3.7	UC07 – Visualizzazione errore colonna mancante	17
3.3.8	UC08 – Avvio del flusso di integrazione	17
3.3.9	UC09 – Consultazione delle esecuzioni	18
3.3.10	UC10 – Impostazione dei filtri	19
3.3.11	UC11 – Reset dei filtri	20
3.3.12	UC12 – Visualizzazione dei risultati di un'esecuzione . .	21
3.3.13	UC13 – Visualizzazione degli errori di validazione	23
3.3.14	UC14 – Correzione degli errori di validazione	26
3.3.15	UC15 – Caricamento definitivo dei dati validati	28
3.3.16	UC16 – Logout	29

3.4	Tabella dei requisiti	29
3.4.1	Requisiti funzionali	30
3.4.1.1	Obbligatori	30
3.4.1.2	Desiderabili	32
3.4.1.3	Opzionali	33
3.4.2	Requisiti di qualità	33
3.4.2.1	Obbligatori	33
3.4.2.2	Desiderabili	34
3.4.3	Requisiti di vincolo	34
3.4.3.1	Obbligatori	34
3.4.4	Riepilogo dei requisiti	35
4	Analisi tecnologica e scelta degli strumenti	36
4.1	Contesto tecnologico	36
4.2	Fondamenti dell'integrazione dei dati	38
4.2.1	Processi ETL ed ELT	38
4.2.2	Flusso di integrazione e orchestrazione	39
4.3	Selezione preliminare degli strumenti	39
4.4	Scenario e criteri del Proof of Concept	41
4.5	Valutazione comparativa delle soluzioni	42
4.5.1	n8n	42
4.5.2	Apache Airflow	43
4.5.3	Apache NiFi	44
4.5.4	Google Cloud Composer	46
4.5.5	Google Cloud Data Fusion	46
4.5.6	Qlik	47
4.5.7	Esito della sperimentazione	49
5	Progettazione della soluzione	50
5.1	Criteri progettuali	50
5.2	Architettura generale del sistema	50
5.3	Tecnologie utilizzate	52
5.3.1	Backend	52
5.3.2	Frontend	54
5.3.3	Database	55
5.3.4	Infrastruttura	55
5.3.5	Strumenti di sviluppo	56
5.4	Progettazione del backend e delle API	57
5.4.1	Architettura logica del backend	57
5.4.2	Diagramma delle classi principali	58

5.4.3	Organizzazione delle API	60
5.5	Progettazione del modello dati	61
5.5.1	Area di staging	61
5.5.2	Tabelle di controllo e tracciabilità	62
5.5.3	Tabelle di frontiera	64
5.6	Progettazione del mapping dei campi	66
5.7	Progettazione del flusso ETL	67
5.7.1	Ricezione del file e creazione della run	67
5.7.2	Controllo del tracciato e normalizzazione	67
5.7.3	Validazione dei dati	68
5.7.4	Correzione e rivalidazione	68
5.7.5	Popolamento delle tabelle di frontiera	69
5.7.6	Caricamento definitivo	69
5.7.7	Ciclo di vita della run	69
5.8	Aspetti trasversali	70
5.8.1	Integrità e gestione degli errori	71
5.8.2	Prestazioni e gestione dei volumi	71
5.8.3	Sicurezza e isolamento dei tenant	71
6	Sviluppo del sistema ETL	72
6.1	Punto di partenza e versioni realizzate	72
6.2	Orchestrazione con n8n	73
6.2.1	Comunicazione tra backend e n8n	74
6.2.2	Workflow di normalizzazione	75
6.2.3	Workflow di validazione e caricamento nelle tabelle di frontiera	76
6.3	Implementazione del backend	77
6.3.1	Struttura del modulo ETL	77
6.3.2	Validazione dei dati	78
6.3.3	Correzione e rivalidazione	79
6.3.4	Popolamento delle tabelle di frontiera	80
6.3.5	Caricamento definitivo	81
6.4	Implementazione del frontend	84
6.4.1	Home del sistema ETL	84
6.4.2	Caricamento del file	85
6.4.3	Consultazione delle esecuzioni	85
6.4.4	Correzione degli errori	86
6.4.5	Caricamento nel database definitivo	87
6.5	Implementazione della versione senza n8n	88
6.5.1	Spostamento della normalizzazione nel backend	88

6.5.2	Gestione del flusso dal backend	89
6.5.3	Coda del caricamento definitivo	90
6.6	Implementazione del mapping assistito dall'intelligenza artificiale	92
6.6.1	Preparazione della richiesta	92
6.6.2	Generazione e controllo del mapping	93
6.6.3	Conferma da parte dell'Utente	94
7	Valutazione della soluzione	96
7.1	Raggiungimento degli obiettivi e verifica dei requisiti	96
7.1.1	Raggiungimento degli obiettivi	96
7.1.2	Verifica dei requisiti	97
7.2	Modalità di valutazione	98
7.3	Valutazione e confronto delle versioni realizzate	98
7.3.1	Correttezza dei risultati	98
7.3.2	Tempi di esecuzione osservati	98
7.3.3	Monitoraggio e gestione degli errori	99
7.3.4	Manutenibilità e semplicità di gestione	99
7.3.5	Valutazione del mapping assistito dall'intelligenza artificiale	100
7.3.6	Valutazione complessiva	100
8	Conclusioni	102
8.1	Limiti e possibili miglioramenti	102
8.1.1	Limiti della soluzione attuale	102
8.1.2	Possibili sviluppi futuri	102
8.2	Esperienza di stage	103
8.2.1	Conoscenze acquisite	103
8.2.2	Considerazioni personali	103
8.3	Considerazioni finali	103
	Sitografia	104
	Acronimi e abbreviazioni	106
	Glossario	108

Elenco delle figure

1.1	Logo di Sinesy	1
3.1	UC01: Autenticazione	10
3.2	UC03: Caricamento del file CSV	12
3.3	UC05: Mappatura dei campi	14
3.4	UC08: Avvio del flusso di integrazione	18
3.5	UC09: Consultazione delle esecuzioni	18
3.6	UC10: Impostazione dei filtri	19
3.7	UC12: Visualizzazione dei risultati di un'esecuzione	21
3.8	UC13: Visualizzazione degli errori di validazione	24
3.9	UC14: Correzione degli errori di validazione	26
3.10	UC15: Caricamento definitivo dei dati validati	28
3.11	UC16: Logout	29
4.1	Macroarchitettura di FourWayShop	37
4.2	Flussi di integrazione esistenti di FourWayShop	38
4.3	Proof of Concept realizzato con n8n	43
4.4	Proof of Concept realizzato con Apache Airflow	44
4.5	Proof of Concept realizzato con Apache NiFi	45
4.6	Pipeline parziale realizzata con Google Cloud Data Fusion	47
4.7	Proof of Concept realizzato con Qlik	48
5.1	Architettura generale del sistema ETL	52
5.2	Architettura logica del backend del sistema ETL	58
5.3	Diagramma delle principali classi e dei moduli del backend ETL	59
5.4	Tabelle di staging, controllo e tracciabilità	63
5.5	Tabelle di frontiera delle entità da caricare in <code>keepit_svil</code>	65
5.6	Tabella utilizzata per memorizzare e riutilizzare i mapping	66
6.1	Workflow n8n per la normalizzazione	75
6.2	Workflow n8n per la validazione e le tabelle di frontiera	76
6.3	Home del sistema ETL	84

6.4	Pagina per il caricamento del file CSV	85
6.5	Elenco delle esecuzioni ETL	86
6.6	Dettaglio di una run con errori di validazione	86
6.7	Procedura guidata per la correzione degli errori	87
6.8	Avanzamento del caricamento nelle tabelle definitive	88
6.9	Caricamento definitivo in coda nella versione senza n8n	92
6.10	Controllo di un'associazione duplicata nel mapping	95
7.1	Esempio di errore mostrato dall'interfaccia di n8n	99

Elenco delle tabelle

2.1	Analisi dei rischi relativi allo svolgimento dello stage e allo sviluppo del progetto.	6
3.1	Campi descrittivi dei casi d'uso	9
3.2	Requisiti funzionali obbligatori	32
3.3	Requisiti funzionali desiderabili	33
3.4	Requisiti funzionali opzionali	33
3.5	Requisiti di qualità obbligatori	34
3.6	Requisiti di qualità desiderabili	34
3.7	Requisiti di vincolo obbligatori	35
3.8	Riepilogo dei requisiti per tipologia e priorità	35
4.1	Prima parte del confronto preliminare degli strumenti	40
4.2	Seconda parte del confronto preliminare degli strumenti	40
5.1	Macroaree dell'architettura del sistema ETL	51
5.2	Tecnologie utilizzate nel backend	53
5.3	Tecnologie utilizzate nel frontend	54
5.4	Database utilizzati nel sistema	55
5.5	Tecnologie e servizi infrastrutturali	56
5.6	Strumenti usati durante lo sviluppo	56

5.7	Principali gruppi di API del modulo ETL	60
5.8	Stati principali della run	70
6.1	Organizzazione delle versioni del progetto	73
7.1	Raggiungimento dei principali obiettivi del progetto	97
7.2	Requisiti svolti rispetto al totale, suddivisi per tipologia e priorità	97
7.3	Confronto tra la versione con n8n e la versione backend	101

Elenco dei codici sorgenti

1.1	Esempio di codice	3
6.1	Controllo delle chiamate interne provenienti da n8n	75
6.2	Esempio di regole applicate a una singola riga	78
6.3	Esecuzione delle regole e aggiornamento dello staging	79
6.4	Creazione delle associazioni tra articoli e fornitori	81
6.5	Ordine delle funzioni di caricamento definitivo	82
6.6	Suddivisione tra record da inserire e da aggiornare	83
6.7	Estratto della normalizzazione di un blocco CSV	89
6.8	Esecuzione del flusso ETL direttamente dal backend	90
6.9	Esecuzione in sequenza dei caricamenti definitivi	91
6.10	Preparazione delle intestazioni e delle righe di esempio	93

Capitolo 1

Introduzione

In questo capitolo viene introdotto il contesto generale in cui si colloca il progetto di stage, presentando l'azienda ospitante e le motivazioni alla base del lavoro svolto. Vengono inoltre descritti gli obiettivi della tesi e l'organizzazione dei capitoli successivi.

1.1 L'azienda

Sinesy S.r.l. è un'azienda informatica con sede a Oderzo, in provincia di Treviso, specializzata nel rendere digitali i processi di lavoro delle aziende medio-grandi attraverso soluzioni software dedicate. Le sue attività comprendono la realizzazione di applicazioni web e mobile, l'integrazione con sistemi informativi esistenti e la digitalizzazione dei processi aziendali.

Uno degli ambiti principali in cui opera l'azienda è il settore *Retail^G*, per il quale propone soluzioni orientate alla gestione dei processi di vendita, magazzino e relazione con il cliente. In questo contesto si colloca la suite proprietaria *4WS^G*, un software gestionale retail pensato per supportare la gestione integrata di negozi fisici ed e-commerce. Le competenze dell'azienda si estendono inoltre anche ai settori Manufacturing, Healthcare e Finance.



Figura 1.1: Logo di Sinesy

1.2 Il progetto

L'idea del progetto nasce dalla necessità di rendere più semplice, controllato e affidabile il processo di integrazione dei dati provenienti dai clienti. La soluzione

precedentemente adottata, infatti, mostrava alcune difficoltà nella gestione di flussi di integrazione sempre più complessi, caratterizzati da elevati volumi di dati, dalla presenza di numerosi *tenant*^G e da un'elevata necessità di manutenzione. In un contesto aziendale in cui le informazioni possono arrivare in formati diversi, è quindi importante disporre di un sistema capace di trasformare dati esterni in informazioni coerenti, verificabili e compatibili con i sistemi interni.

Il progetto prevede inizialmente l'analisi dei principali strumenti utili alla realizzazione di flussi di integrazione ed elaborazione dati, con l'obiettivo di individuare le soluzioni più adatte rispetto alle esigenze aziendali. A partire da questa valutazione, il lavoro mira poi a definire una soluzione *ETL*^G capace di guidare l'intero processo di importazione: dall'acquisizione del file alla *normalizzazione*^G dei dati, dalla validazione alla gestione degli errori, fino alla preparazione delle informazioni per il caricamento nel gestionale.

L'importanza della soluzione risiede nella possibilità di ridurre le attività manuali, limitare il rischio di errori, migliorare la tracciabilità delle elaborazioni e garantire una maggiore qualità dei dati aziendali.

1.3 Organizzazione del testo

Il documento si divide in otto capitoli principali, inclusa questa introduzione.

Il secondo capitolo descrive il contesto dello stage, l'organizzazione dell'attività, il rapporto con l'azienda ospitante e la metodologia di lavoro adottata.

Il terzo capitolo approfondisce l'analisi dei requisiti del progetto, individuando il problema da risolvere, gli attori coinvolti, i principali casi d'uso e i requisiti funzionali e non funzionali.

Il quarto capitolo descrive il confronto tra diversi strumenti ETL, dalla raccolta iniziale di informazioni alla realizzazione di *PoC*^G utili a individuare la soluzione più adatta al progetto.

Il quinto capitolo descrive la progettazione della soluzione, illustrando l'architettura generale del sistema, la separazione delle responsabilità tra i componenti, il flusso applicativo e il modello dati previsto.

Il sesto capitolo approfondisce lo sviluppo del sistema ETL, descrivendo le implementazioni realizzate, il ruolo di *n8n*^G, la logica sviluppata nel *backend*^G, l'interfaccia *frontend*^G e il modulo di *mapping*^G assistito dall'intelligenza artificiale.

Il settimo capitolo presenta la valutazione della soluzione realizzata, riportando le attività di test e il confronto rispetto ai requisiti iniziali.

L'ottavo capitolo descrive i limiti individuati, i possibili sviluppi futuri, le conoscenze acquisite e le considerazioni conclusive.

Per la stesura del documento sono state adottate le seguenti convenzioni tipografiche:

- gli acronimi, le abbreviazioni e i termini ambigui o di uso non comune menzionati vengono definiti nel glossario, situato alla fine del presente documento; la prima occorrenza di ciascun termine è contrassegnata da una G in apice;
- i nomi di funzioni o variabili appartenenti a un linguaggio di programmazione vengono scritti con un carattere **monospaziato**;
- i termini in lingua straniera o appartenenti al gergo tecnico possono essere evidenziati con il carattere *corsivo* quando ciò ne facilita la comprensione;
- i riferimenti alle fonti online sono indicati con un numero tra parentesi quadre, ad esempio [1], che rimanda alla corrispondente voce della sitografia;
- i blocchi di codice sono rappresentati nel seguente modo:

```
1  #include <stdio.h>
2  int main(void) {
3      printf("Hello, world!\n");
4      return 0;
5  }
```

Codice 1.1: Esempio di codice

Capitolo 2

Descrizione dello stage

2.1 Contesto aziendale e organizzativo

Lo stage si è svolto presso Sinesy S.r.l. e ha riguardato un progetto pensato per essere integrato all'interno di FourWayShop, la suite software proprietaria dell'azienda. Il lavoro si colloca quindi in un contesto applicativo reale, legato all'evoluzione di un prodotto già esistente e alla necessità di migliorare la gestione dei flussi di integrazione dati provenienti dai clienti.

Durante lo svolgimento dello stage, l'attività è stata seguita dal tutor aziendale, responsabile dell'area Ricerca e Sviluppo.

2.2 Pianificazione delle attività

Il piano di lavoro dello stage prevedeva una durata complessiva di 304 ore, distribuite in otto settimane.

1. Introduzione in azienda e raccolta requisiti – 40 ore

Comprendeva il setup tecnico dell'ambiente di lavoro, l'introduzione ai processi aziendali, la lettura della documentazione disponibile, l'allineamento sugli obiettivi finali e la raccolta dei requisiti.

2. Studio e sperimentazione – 80 ore

Comprendeva la formazione autonoma sugli strumenti e sulle tecnologie utili alla realizzazione di flussi ETL/ELT, con prime attività di sperimentazione tecnica.

3. Definizione dell'architettura, sviluppo e collaudo del Proof of Concept – 80 ore

Comprendeva la definizione dell'architettura di riferimento, l'implementazione di un Proof of Concept su un caso d'uso reale, il collaudo della soluzione, la misurazione delle performance e la revisione dei risultati sulla base dei feedback ricevuti.

4. Esplorazione di funzionalità basate su intelligenza artificiale – 80 ore

Comprendeva l'analisi e la valutazione di possibili funzionalità di intelligenza artificiale applicabili a supporto del processo di integrazione dati.

5. Report e condivisione dei risultati – 24 ore

Comprendeva il completamento e la rifinitura del progetto, la produzione della documentazione finale, la preparazione della presentazione dei risultati, la condivisione con il team operativo e il colloquio conclusivo di feedback.

2.3 Metodo di lavoro adottato

Il lavoro è stato organizzato attraverso un confronto quotidiano con il tutor aziendale, che indicava le attività da svolgere e le priorità da seguire. Al termine delle singole attività, veniva effettuato un nuovo confronto per verificare il risultato ottenuto, chiarire eventuali dubbi e definire i passi successivi.

Durante lo stage si sono tenuti anche momenti di condivisione con il team di sviluppo e con il responsabile del team Delivery. Questi incontri hanno permesso di raccogliere feedback sul lavoro svolto, valutare eventuali modifiche e verificare che la soluzione fosse coerente sia con le esigenze tecniche sia con quelle operative dell'azienda.

2.4 Analisi dei rischi

L'analisi dei rischi è stata svolta con lo scopo di individuare le principali criticità che avrebbero potuto influenzare lo svolgimento dello stage e il raggiungimento degli obiettivi previsti. Per ciascun rischio sono stati indicati un codice identificativo, una breve descrizione, la strategia di mitigazione, la probabilità di occorrenza e l'impatto sul progetto.

Codice	Nome	Descrizione	Mitigazione	Probabilità	Impatto
R01	Requisiti non completamente definiti	Alcuni requisiti avrebbero potuto emergere o cambiare durante lo stage, in base al confronto con i team coinvolti.	Mantenere confronti frequenti con il tutor e validare progressivamente le scelte.	Media	Medio
R02	Scelta tecnologica non adeguata	Lo strumento ETL scelto avrebbe potuto non soddisfare pienamente esigenze di funzionalità, integrazione o manutenibilità.	Confrontare più soluzioni e realizzare prove pratiche prima della scelta finale.	Media	Alto
R03	Complessità dei dati in ingresso	I file dei clienti avrebbero potuto presentare strutture diverse, campi mancanti, valori incoerenti o dati non conformi.	Prevedere normalizzazione, validazione e registrazione degli errori rilevati.	Alta	Alto
R04	Gestione di file di grandi dimensioni	File con molte righe avrebbero potuto causare problemi di performance, tempi elevati o consumo eccessivo di risorse.	Testare dataset crescenti e adottare strategie di elaborazione a blocchi.	Media	Alto
R05	Integrazione con sistemi esistenti	La soluzione avrebbe potuto incontrare difficoltà nell'interazione con database, API^G o componenti aziendali.	Analizzare i vincoli esistenti e validare l'integrazione con il team tecnico.	Media	Medio
R06	Monitoraggio e gestione degli errori insufficienti	Una gestione poco chiara degli errori avrebbe potuto rendere difficile comprendere lo stato e le cause dei problemi.	Progettare tracciamento delle run^G , salvataggio errori e consultazione degli esiti.	Media	Alto
R07	Tempi limitati dello stage	Il tempo disponibile avrebbe potuto limitare funzionalità e approfondimenti tecnici.	Dare priorità agli obiettivi principali, procedere per incrementi e documentare sviluppi futuri.	Media	Medio
R08	Difficoltà nell'uso di strumenti nuovi	Tecnologie non conosciute avrebbero potuto rallentare le prime fasi di sviluppo e sperimentazione.	Dedicare tempo iniziale alla formazione, alla documentazione e al confronto con il tutor.	Media	Medio

Tabella 2.1: Analisi dei rischi relativi allo svolgimento dello stage e allo sviluppo del progetto.

Capitolo 3

Analisi dei requisiti

Questo capitolo presenta l'analisi dei requisiti del progetto, descrivendo il problema da risolvere e gli obiettivi della soluzione proposta. Successivamente vengono illustrati gli attori coinvolti, i principali casi d'uso e i requisiti funzionali, di qualità e di vincolo individuati.

3.1 Problema da risolvere e obiettivi

FourWayShop riceve periodicamente dalle aziende clienti del settore Retail i dati relativi alle diverse anagrafiche da inserire o aggiornare all'interno dei sistemi aziendali. Tali dati comprendono, ad esempio, le informazioni riguardanti prodotti, varianti, categorie e altre entità necessarie al funzionamento della piattaforma.

I clienti trasmettono queste informazioni attraverso file *CSV*^G. Sebbene i dati contenuti siano generalmente simili, ogni azienda adotta una propria struttura, una diversa nomenclatura dei campi e specifiche modalità di rappresentazione dei valori. Non è quindi disponibile un formato standard comune e ogni importazione richiede operazioni di analisi, adattamento e associazione tra le colonne del file e i campi presenti nel database di FourWayShop.

La soluzione utilizzata inizialmente dall'azienda era stata sviluppata diversi anni prima e risultava complessa da gestire, poco flessibile e difficilmente manutenibile. Inoltre, non era progettata per gestire in modo efficiente flussi di integrazione sempre più complessi, grandi volumi di dati e un numero crescente di clienti (tenant). L'aggiunta di nuovi clienti o la modifica di un tracciato esistente poteva richiedere interventi specifici sul sistema, aumentando il tempo necessario per preparare le importazioni e i relativi costi di manutenzione.

Un'ulteriore criticità riguardava la qualità dei dati forniti. I file potevano contenere valori mancanti, formati non validi, riferimenti errati o informazioni incompatibili con i vincoli e le dipendenze presenti nel database. In assenza di

un processo strutturato di validazione, tali anomalie potevano causare errori durante l'importazione o compromettere la coerenza dei dati già presenti nel sistema.

Il problema da risolvere consiste quindi nel rendere più rapido, sicuro e flessibile il processo di inserimento e aggiornamento delle anagrafiche, riducendo le attività manuali e la dipendenza da interventi tecnici specifici. La soluzione deve inoltre permettere di riutilizzare i flussi di importazione, gestire strutture CSV differenti, supportare l'elaborazione di grandi volumi di dati, operare in un contesto multi-tenant e monitorare le elaborazioni.

L'obiettivo del progetto è la realizzazione di un flusso ETL accessibile attraverso un portale dedicato, mediante il quale un operatore possa caricare autonomamente un file CSV, configurare l'associazione tra i dati ricevuti e quelli presenti nel sistema, verificarne la correttezza e procedere con il caricamento nel database. Prima dell'importazione definitiva, il sistema deve individuare e segnalare eventuali errori, consentendo all'operatore di correggerli direttamente attraverso l'interfaccia.

Il processo deve risultare sufficientemente semplice da poter essere utilizzato anche da operatori privi di competenze tecniche specifiche, garantendo allo stesso tempo l'integrità delle relazioni e delle dipendenze tra i dati. È inoltre necessario mantenere la tracciabilità delle operazioni eseguite, così da agevolare il monitoraggio del flusso e l'individuazione delle cause di eventuali anomalie. La soluzione proposta mira pertanto ad automatizzare il processo di integrazione dei dati migliorandone flessibilità, affidabilità e tracciabilità.

3.2 Attori

Il sistema adotta un modello multi-tenant, nel quale ogni cliente dispone di uno spazio isolato e non può accedere alle risorse degli altri tenant.

Nel modello dei casi d'uso è presente un unico attore, denominato Utente, che rappresenta chiunque utilizzi la piattaforma tramite l'interfaccia web. L'Utente può consultare le proprie integrazioni, caricare i file di input, avviare le esecuzioni e verificarne lo stato e gli eventuali errori.

Per scelta operativa, Sinesy ha deciso di non implementare ruoli e funzionalità differenti per clienti e amministratori. Le attività di configurazione delle integrazioni e gestione degli utenti vengono svolte direttamente in FourWayShop, attraverso una sezione già dedicata a tali operazioni. Quando un operatore Sinesy accede alla piattaforma, utilizza quindi le credenziali dello specifico tenant e dispone delle stesse funzionalità del cliente.

3.3 Casi d'uso

Ogni caso d'uso è descritto attraverso le informazioni riportate nella seguente tabella, fatta eccezione per i campi non pertinenti.

Campo	Descrizione
Grafico <i>UML</i> ^G	Rappresenta lo scenario dei casi d'uso in oggetto.
Attore	Rappresenta coloro che interagiscono con il sistema dall'esterno, senza farne parte.
Scenario principale	La sequenza ragionevole delle operazioni che l'attore deve effettuare per portare a compimento lo scenario.
Precondizioni	Lista di elementi necessari affinché l'attore possa soddisfare il caso d'uso in oggetto.
Postcondizioni	Lista di elementi che descrivono le modifiche effettuate internamente dopo il completamento dello scenario principale.
Scenario alternativo	Rappresenta un comportamento valido ma non principale, che devia dal flusso base al verificarsi di una condizione specifica.
Inclusioni	Ulteriori casi d'uso che l'attore deve compiere per portare a termine lo scenario.
Estensioni	Relazione che aggiunge un comportamento opzionale o alternativo a un caso d'uso completo, attivata solo al verificarsi di una condizione specifica.
Specializzazioni	Le specializzazioni rappresentano varianti del caso d'uso generale e sono tra loro mutualmente esclusive, senza imporre un percorso di esecuzione obbligatorio.
Trigger	Evento o condizione che avvia il caso d'uso.

Tabella 3.1: Campi descrittivi dei casi d'uso

3.3.1 UC01 – Autenticazione

UC01: Autenticazione

Grafico UML:

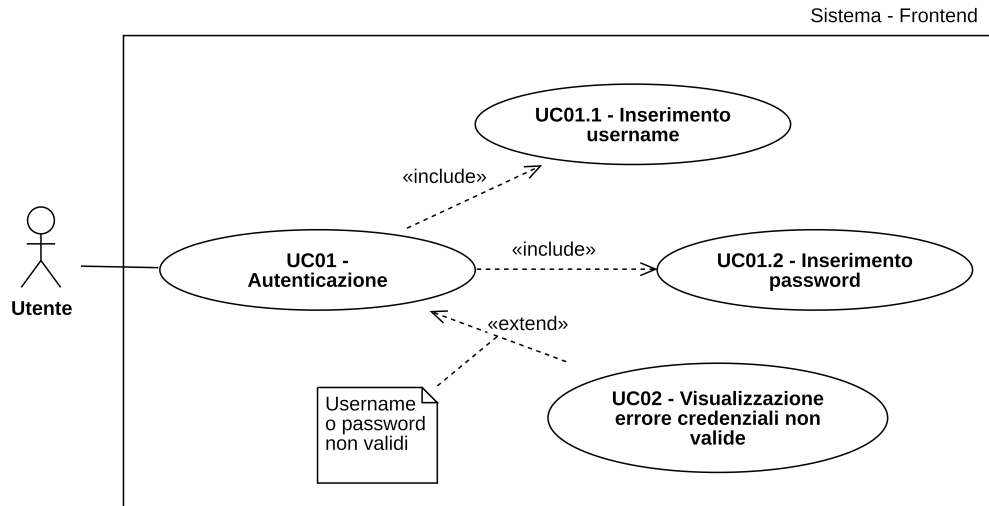


Figura 3.1: UC01: Autenticazione

Attore: Utente.

Scenario principale: L'Utente inserisce username e password e invia la richiesta di accesso. Il sistema verifica le credenziali, identifica il tenant e apre la pagina iniziale della piattaforma.

Precondizioni: L'Utente non è già autenticato e visualizza la pagina di accesso.

Postcondizioni: È creata una sessione autenticata associata all'Utente e al relativo tenant.

Scenario alternativo: Se le credenziali inserite non sono valide, il sistema esegue UC02.

Inclusioni: UC01.1 – Inserimento username; UC01.2 – Inserimento password.

Estensioni: UC02 – Visualizzazione errore credenziali non valide.

Trigger: L'Utente richiede di accedere alla piattaforma.

UC01.1: Inserimento username

Attore: Utente.

Scenario principale: L'Utente digita lo username nel campo previsto; il sistema acquisisce il valore inserito.

Precondizioni: La pagina di autenticazione è visualizzata.

Postcondizioni: Lo username è disponibile per la successiva richiesta di autenticazione.

Trigger: L'Utente seleziona il campo dedicato allo username nella pagina di accesso.

UC01.2: Inserimento password

Attore: Utente.

Scenario principale: L'Utente digita la password nel campo previsto; il sistema acquisisce il valore mascherandone i caratteri.

Precondizioni: La pagina di autenticazione è visualizzata.

Postcondizioni: La password è disponibile per la successiva richiesta di autenticazione.

Trigger: L'Utente seleziona il campo dedicato alla password nella pagina di accesso.

3.3.2 UC02 – Visualizzazione errore credenziali non valide

UC02: Visualizzazione errore credenziali non valide

Attore: Utente.

Scenario principale: Il sistema rifiuta la richiesta, mantiene l'Utente nella pagina di accesso e visualizza un messaggio di errore senza specificare quale credenziale sia errata.

Precondizioni: L'Utente ha inviato credenziali non valide.

Postcondizioni: Nessuna sessione autenticata viene creata; l'Utente può inserire nuovamente le credenziali.

Trigger: La verifica delle credenziali inserite durante UC01 restituisce esito negativo.

3.3.3 UC03 – Caricamento del file CSV

UC03: Caricamento del file CSV

Grafico UML:

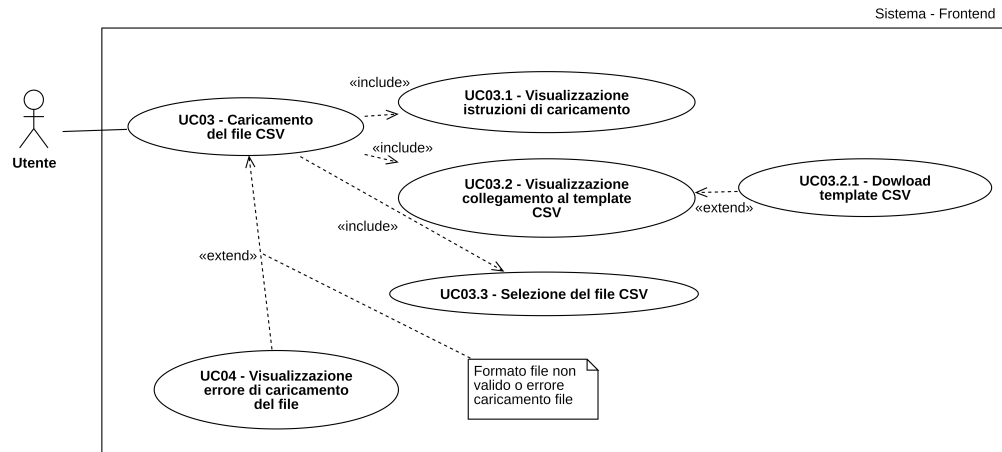


Figura 3.2: UC03: Caricamento del file CSV

Attore: Utente.

Scenario principale: Il sistema mostra le istruzioni e il collegamento al template; l'Utente seleziona un file CSV e conferma il caricamento; il sistema verifica il file e lo rende disponibile per l'eventuale mappatura dei campi o per l'avvio del flusso.

Precondizioni: L'Utente è autenticato e dispone di un'integrazione configurata per il proprio tenant.

Postcondizioni: Il file è stato verificato ed è disponibile per l'eventuale mappatura dei campi o per l'avvio del flusso.

Scenario alternativo: Se il caricamento fallisce, il sistema esegue UC04.

Inclusioni: UC03.1 – Visualizzazione delle istruzioni di caricamento; UC03.2 – Visualizzazione del collegamento al template CSV; UC03.3 – Selezione del file CSV.

Estensioni: UC04 – Visualizzazione errore di caricamento del file.

Trigger: L'Utente avvia una nuova importazione.

UC03.1: Visualizzazione delle istruzioni di caricamento

Attore: Utente.

Scenario principale: Il sistema visualizza le istruzioni relative al formato CSV, alla struttura attesa e alla modalità di selezione del file.

Precondizioni: L'Utente è autenticato.

Postcondizioni: L'Utente dispone delle informazioni necessarie per scegliere un file compatibile.

Trigger: L'Utente apre la pagina per una nuova importazione.

UC03.2: Visualizzazione del collegamento al template CSV

Attore: Utente.

Scenario principale: Il sistema mostra nella pagina di caricamento il collegamento al template CSV associato all'integrazione.

Precondizioni: L'Utente è autenticato e la pagina di caricamento è visualizzata.

Postcondizioni: Il collegamento al template CSV è visibile e può essere selezionato dall'Utente.

Scenario alternativo: L'Utente può scaricare il template tramite UC03.2.1.

Estensioni: UC03.2.1 – Download del template CSV.

Trigger: L'Utente apre la pagina dedicata al caricamento del file CSV.

UC03.2.1: Download del template CSV

Attore: Utente.

Scenario principale: L'Utente seleziona il collegamento; il sistema recupera il template associato all'integrazione e ne avvia il download.

Precondizioni: Il collegamento al template CSV è visualizzato.

Postcondizioni: Il template CSV è disponibile sul dispositivo dell'Utente.

Trigger: L'Utente seleziona il collegamento al template CSV.

UC03.3: Selezione del file CSV

Attore: Utente.

Scenario principale: Il sistema apre il selettore; l'Utente sceglie un file; il sistema ne mostra il nome e lo prepara per il caricamento.

Precondizioni: La pagina di caricamento è visualizzata.

Postcondizioni: Il file scelto è associato al modulo di caricamento.

Trigger: L'Utente attiva il controllo di selezione del file.

3.3.4 UC04 – Visualizzazione errore di caricamento del file

UC04: Visualizzazione errore di caricamento del file

Attore: Utente.

Scenario principale: Il sistema interrompe il caricamento, visualizza un messaggio comprensibile e consente all'Utente di scegliere un altro file.

Grafico UML:



Trigger: L'analisi del file rileva intestazioni non conformi e non trova un mapping già utilizzabile.

UC05.1: Visualizzazione della mappatura proposta

Attore: Utente.

Scenario principale: Per ogni colonna il sistema mostra il nome sorgente, la corrispondenza individuata, alcuni valori di esempio e lo stato della mappatura.

Precondizioni: Il sistema ha generato una nuova proposta di mappatura.

Postcondizioni: La proposta è visibile e può essere verificata o modificata dall'Utente.

Inclusioni: UC05.1.1, UC05.1.2, UC05.1.3 e UC05.1.4.

Trigger: La nuova proposta di mappatura è disponibile.

UC05.1.1: Visualizzazione dei campi del file

Attore: Utente.

Scenario principale: Il sistema elenca i nomi delle colonne mantenendo un riferimento univoco alla loro posizione nel file.

Precondizioni: Il sistema ha letto l'intestazione del file CSV.

Postcondizioni: L'Utente può identificare tutti i campi disponibili nel file.

Trigger: Viene aperta la proposta di mappatura.

UC05.1.2: Visualizzazione delle corrispondenze individuate

Attore: Utente.

Scenario principale: Accanto a ciascuna colonna sorgente viene visualizzato il campo di destinazione proposto oppure l'assenza di una corrispondenza.

Precondizioni: Il sistema ha confrontato le colonne sorgenti con i campi di destinazione.

Postcondizioni: L'Utente può valutare la correttezza delle corrispondenze.

Trigger: Viene aperta la proposta di mappatura.

UC05.1.3: Visualizzazione dei valori di esempio

Attore: Utente.

Scenario principale: Per ogni campo sorgente il sistema recupera e visualizza alcuni valori rappresentativi.

Precondizioni: Il file contiene almeno una riga di dati leggibile.

Postcondizioni: L'Utente dispone di un supporto informativo per verificare la mappatura.

Trigger: Viene aperta la proposta di mappatura.

UC05.1.4: Visualizzazione dei campi mappati

Attore: Utente.

Scenario principale: Il sistema aggiorna lo stato grafico delle righe e indica quali campi risultano correttamente mappati.

Precondizioni: Esiste almeno un'associazione tra una colonna sorgente e un campo di destinazione.

Postcondizioni: Lo stato corrente della mappatura è immediatamente riconoscibile.

Trigger: Viene aperta o modificata la proposta di mappatura.

UC05.2: Modifica della mappatura di un campo

Attore: Utente.

Scenario principale: Il sistema mostra i campi disponibili; l'Utente ne seleziona uno; il sistema aggiorna la proposta e ne verifica la coerenza.

Precondizioni: La proposta di mappatura è visualizzata e non è ancora stata confermata.

Postcondizioni: La proposta contiene la nuova associazione scelta dall'Utente.

Scenario alternativo: Se due o più colonne sono associate allo stesso campo, il sistema esegue UC06.

Estensioni: UC06 – Visualizzazione errore colonna duplicata.

Trigger: L'Utente seleziona l'associazione relativa a una colonna.

UC05.3: Conferma della mappatura

Attore: Utente.

Scenario principale: Il sistema valida l'insieme delle associazioni, le salva e abilita l'avvio dell'integrazione.

Precondizioni: La proposta di mappatura è visualizzata e l'Utente può richiederne la conferma.

Postcondizioni: La mappatura viene salvata e può essere usata dal flusso o riutilizzata per altri file con la stessa struttura.

Scenario alternativo: Se una o più colonne obbligatorie non sono state associate, il sistema esegue UC07. Se due o più colonne sono associate allo stesso campo, il sistema esegue UC06. In entrambi i casi la mappatura non viene confermata.

Estensioni: UC06 – Visualizzazione errore colonna duplicata; UC07 – Visualizzazione errore colonna mancante.

Trigger: L'Utente seleziona il comando di conferma.

UC05.4: Annullamento della mappatura

Attore: Utente.

Scenario principale: Il sistema richiede conferma dell'annullamento; l'Utente conferma; il sistema abbandona la proposta corrente.

Precondizioni: La mappatura è in corso e non è stata confermata.

Postcondizioni: Le modifiche non confermate non sono utilizzate dal flusso.

Trigger: L'Utente seleziona il comando di annullamento.

3.3.6 UC06 – Visualizzazione errore colonna duplicata

UC06: Visualizzazione errore colonna duplicata

Attore: Utente.

Scenario principale: Il sistema evidenzia tutte le associazioni in conflitto e invita l'Utente a mantenere una sola corrispondenza valida.

Precondizioni: L'Utente sta modificando la mappatura oppure ne ha richiesto la conferma.

Postcondizioni: La mappatura rimane non confermabile finché il conflitto non viene risolto.

Trigger: Due o più colonne sono state associate allo stesso campo di destinazione.

3.3.7 UC07 – Visualizzazione errore colonna mancante

UC07: Visualizzazione errore colonna mancante

Attore: Utente.

Scenario principale: Il sistema identifica il campo richiesto, evidenzia la mancanza e invita l'Utente a selezionare una colonna compatibile.

Precondizioni: L'Utente ha richiesto la conferma della mappatura.

Postcondizioni: La mappatura rimane non confermabile finché il campo obbligatorio non viene associato.

Trigger: Una o più colonne obbligatorie non sono state associate.

3.3.8 UC08 – Avvio del flusso di integrazione

UC08: Avvio del flusso di integrazione

Grafico UML:

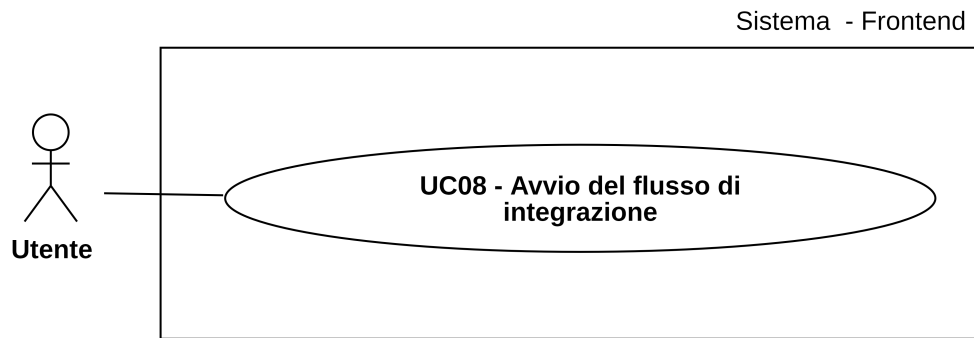


Figura 3.4: UC08: Avvio del flusso di integrazione

Attore: Utente.

Scenario principale: Il sistema crea l'esecuzione, ne mostra lo stato iniziale e avvia il flusso, che normalizza e valida i dati distinguendo le righe valide da quelle non valide.

Precondizioni: Il file CSV è stato caricato e presenta intestazioni già conformi oppure la relativa mappatura è stata confermata.

Postcondizioni: È presente una nuova esecuzione tracciata e associata al file e al tenant dell'Utente.

Trigger: L'Utente seleziona il comando per avviare l'elaborazione.

3.3.9 UC09 – Consultazione delle esecuzioni

UC09: Consultazione delle esecuzioni

Grafico UML:

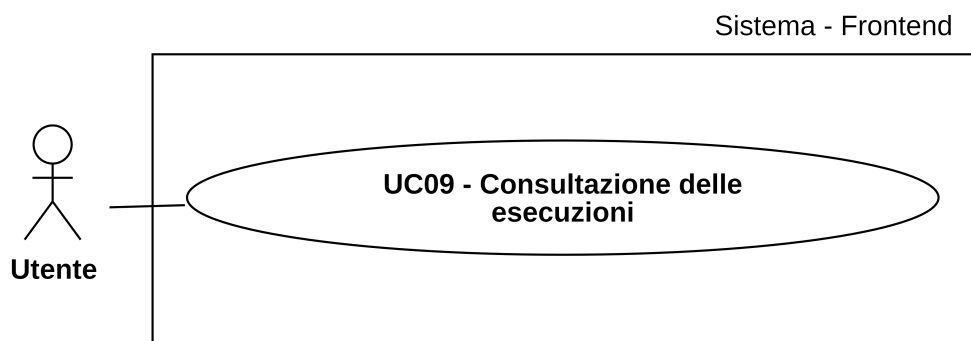


Figura 3.5: UC09: Consultazione delle esecuzioni

Attore: Utente.

Scenario principale: Il sistema carica l'elenco delle esecuzioni associate al tenant; l'Utente può scorrerlo e individuare l'esecuzione di interesse.

Precondizioni: L'Utente è autenticato.

Postcondizioni: L'elenco corrente rimane disponibile per la selezione di un'esecuzione.

Trigger: L'Utente apre la sezione dedicata alle esecuzioni.

3.3.10 UC10 – Impostazione dei filtri

UC10: Impostazione dei filtri

Grafico UML:

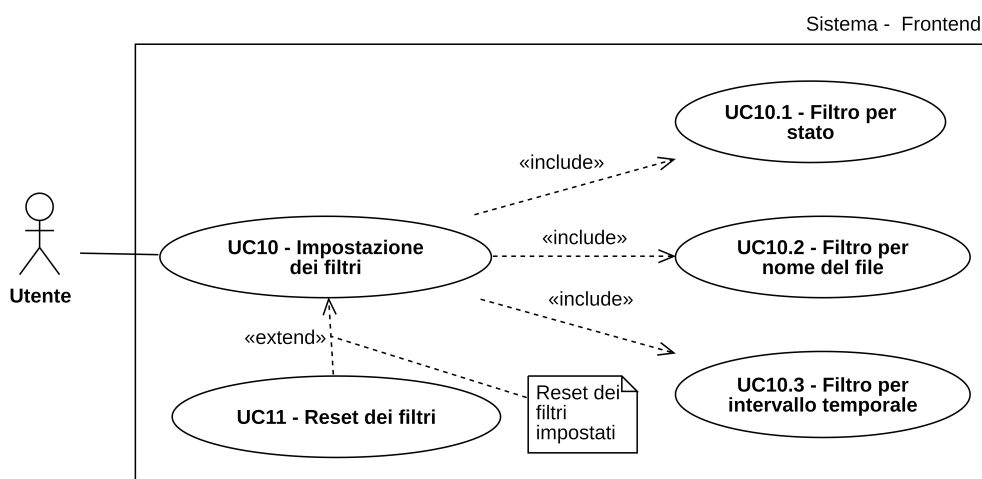


Figura 3.6: UC10: Impostazione dei filtri

Attore: Utente.

Scenario principale: L'Utente valorizza i filtri desiderati; il sistema combina i criteri e aggiorna l'elenco.

Precondizioni: L'elenco delle esecuzioni è visualizzato.

Postcondizioni: Sono visualizzate soltanto le esecuzioni che soddisfano i filtri applicati.

Scenario alternativo: L'Utente può azzerare i filtri e ripristinare l'elenco completo mediante UC11.

Inclusioni: UC10.1 – Filtro per stato; UC10.2 – Filtro per nome del file; UC10.3 – Filtro per intervallo temporale.

Estensioni: UC11 – Reset dei filtri.

Trigger: L'Utente apre o modifica i controlli di filtro.

UC10.1: Filtro per stato

Attore: Utente.

Scenario principale: Il sistema applica gli stati selezionati e aggiorna l'elenco con le sole esecuzioni corrispondenti.

Precondizioni: La funzione di filtro delle esecuzioni è disponibile.

Postcondizioni: L'elenco rispetta il criterio di stato impostato.

Trigger: L'Utente seleziona uno o più stati nel filtro dedicato.

UC10.2: Filtro per nome del file

Attore: Utente.

Scenario principale: Il sistema confronta il testo inserito con i nomi dei file e aggiorna l'elenco con le corrispondenze.

Precondizioni: La funzione di filtro delle esecuzioni è disponibile.

Postcondizioni: L'elenco rispetta il criterio sul nome del file.

Trigger: L'Utente inserisce un testo nel filtro per nome del file.

UC10.3: Filtro per intervallo temporale

Attore: Utente.

Scenario principale: Il sistema valida le date, applica gli estremi indicati e aggiorna l'elenco.

Precondizioni: La funzione di filtro delle esecuzioni è disponibile.

Postcondizioni: L'elenco contiene soltanto esecuzioni comprese nell'intervallo specificato.

Trigger: L'Utente imposta una data iniziale, una data finale o entrambe.

3.3.11 UC11 – Reset dei filtri

UC11: Reset dei filtri

Attore: Utente.

Scenario principale: L'Utente seleziona il comando di reset; il sistema rimuove tutti i criteri applicati e ripristina l'elenco completo delle esecuzioni.

Precondizioni: L'elenco delle esecuzioni è visualizzato ed è attivo almeno un

filtro.

Postcondizioni: Nessun filtro è attivo e l'elenco mostra tutte le esecuzioni accessibili all'Utente.

Trigger: L'Utente seleziona il comando per reimpostare i filtri.

3.3.12 UC12 – Visualizzazione dei risultati di un'esecuzione

UC12: Visualizzazione dei risultati di un'esecuzione

Grafico UML:

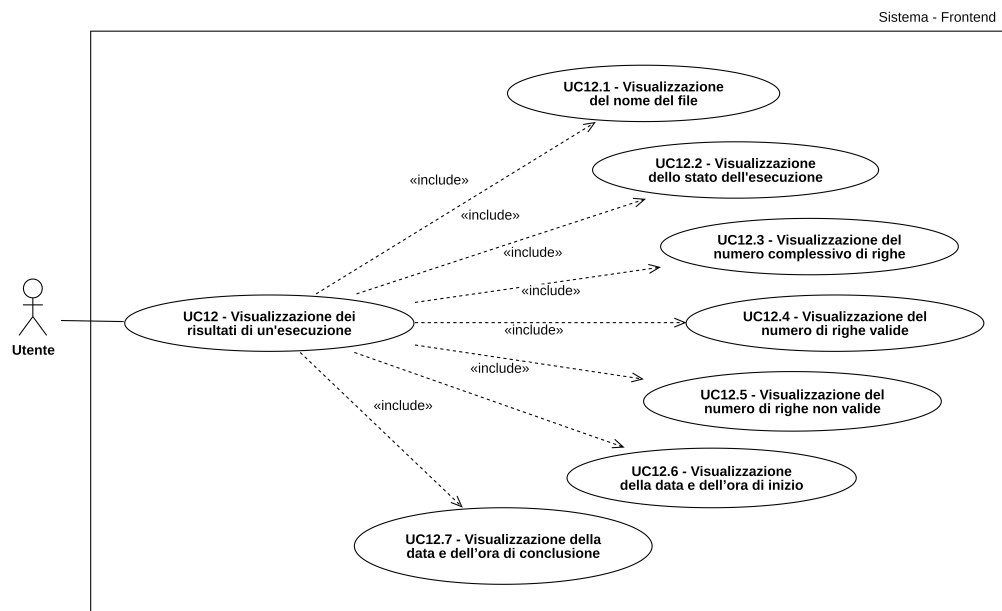


Figura 3.7: UC12: Visualizzazione dei risultati di un'esecuzione

Attore: Utente.

Scenario principale: Il sistema recupera le informazioni dell'esecuzione selezionata e ne visualizza il dettaglio; finché l'elaborazione è in corso, il cruscotto si aggiorna automaticamente al completamento dei diversi passaggi e mostra l'avanzamento dell'esecuzione.

Precondizioni: L'esecuzione appartiene al tenant dell'Utente.

Postcondizioni: I risultati e l'avanzamento dell'esecuzione sono disponibili per la consultazione e restano aggiornati rispetto allo stato dell'elaborazione.

Inclusioni: UC12.1 – Visualizzazione del nome del file; UC12.2 – Visualizzazione dello stato dell'esecuzione; UC12.3 – Visualizzazione del numero complessivo

di righe; UC12.4 – Visualizzazione del numero di righe valide; UC12.5 – Visualizzazione del numero di righe non valide; UC12.6 – Visualizzazione della data e dell'ora di inizio; UC12.7 – Visualizzazione della data e dell'ora di conclusione.

Trigger: L'Utente seleziona un'esecuzione dall'elenco.

UC12.1: Visualizzazione del nome del file

Attore: Utente.

Scenario principale: Il sistema recupera e visualizza il nome originale del file associato all'esecuzione.

Precondizioni: L'esecuzione è associata a un file di input.

Postcondizioni: Il file di origine dell'esecuzione è identificabile.

Trigger: Viene visualizzato il dettaglio dell'esecuzione.

UC12.2: Visualizzazione dello stato dell'esecuzione

Attore: Utente.

Scenario principale: Il sistema recupera lo stato più recente e lo presenta mediante un'etichetta riconoscibile; quando viene completato un passaggio dell'elaborazione, aggiorna automaticamente lo stato mostrato nel cruscotto.

Precondizioni: L'esecuzione esiste.

Postcondizioni: L'Utente conosce lo stato corrente dell'elaborazione.

Trigger: Viene visualizzato il dettaglio dell'esecuzione oppure cambia lo stato dell'elaborazione.

UC12.3: Visualizzazione del numero complessivo di righe

Attore: Utente.

Scenario principale: Il sistema recupera il conteggio totale e lo visualizza nel riepilogo.

Precondizioni: Il sistema ha conteggiato le righe del file.

Postcondizioni: L'Utente conosce la dimensione complessiva dell'elaborazione.

Trigger: Viene visualizzato il dettaglio dell'esecuzione.

UC12.4: Visualizzazione del numero di righe valide

Attore: Utente.

Scenario principale: Il sistema recupera il conteggio delle righe valide e lo visualizza nel riepilogo.

Precondizioni: La validazione ha elaborato almeno una riga.

Postcondizioni: L'Utente conosce il numero di record utilizzabili.

Trigger: Sono disponibili i risultati della validazione.

UC12.5: Visualizzazione del numero di righe non valide

Attore: Utente.

Scenario principale: Il sistema recupera il conteggio delle righe non valide e lo visualizza nel riepilogo.

Precondizioni: La validazione ha elaborato almeno una riga.

Postcondizioni: L'Utente conosce la quantità di record che richiede attenzione.

Trigger: Sono disponibili i risultati della validazione.

UC12.6: Visualizzazione della data e dell'ora di inizio

Attore: Utente.

Scenario principale: Il sistema recupera l'istante di avvio e lo visualizza nel fuso orario previsto dall'interfaccia.

Precondizioni: L'esecuzione è stata avviata.

Postcondizioni: L'Utente conosce la data e l'ora di inizio.

Trigger: Viene visualizzato il dettaglio dell'esecuzione.

UC12.7: Visualizzazione della data e dell'ora di conclusione

Attore: Utente.

Scenario principale: Se l'esecuzione ha raggiunto uno stato conclusivo, il sistema recupera l'istante di conclusione e lo visualizza nel fuso orario previsto dall'interfaccia; in caso contrario, indica che l'informazione non è ancora disponibile.

Precondizioni: L'esecuzione esiste.

Postcondizioni: L'Utente conosce la data e l'ora di conclusione oppure è informato che l'esecuzione non è ancora terminata.

Trigger: Viene visualizzato il dettaglio dell'esecuzione.

3.3.13 UC13 – Visualizzazione degli errori di validazione

UC13: Visualizzazione degli errori di validazione

Grafico UML:

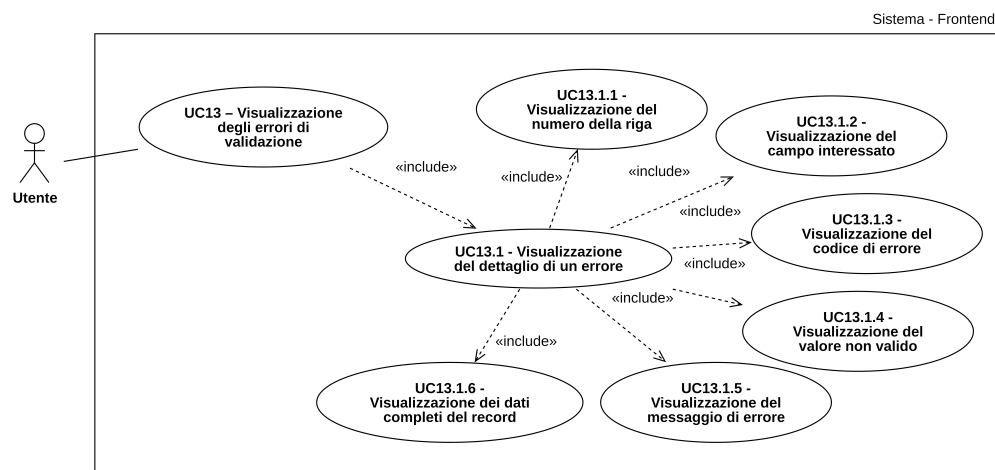


Figura 3.8: UC13: Visualizzazione degli errori di validazione

Attore: Utente.

Scenario principale: Il sistema mostra l'elenco degli errori; l'Utente può selezionarne uno per visualizzarne il dettaglio.

Precondizioni: La validazione ha rilevato almeno un errore e l'esecuzione appartiene al tenant dell'Utente.

Postcondizioni: Gli errori sono disponibili per l'analisi e per l'eventuale correzione.

Inclusioni: UC13.1 – Visualizzazione del dettaglio di un errore.

Trigger: L'Utente apre gli errori di un'esecuzione che contiene righe non valide.

UC13.1: Visualizzazione del dettaglio di un errore

Attore: Utente.

Scenario principale: Il sistema visualizza riga, campo, codice, valore non valido, messaggio e dati completi del record.

Precondizioni: L'errore appartiene all'esecuzione consultata.

Postcondizioni: L'Utente dispone degli elementi necessari per valutare o correggere l'errore.

Inclusioni: UC13.1.1, UC13.1.2, UC13.1.3, UC13.1.4, UC13.1.5 e UC13.1.6.

Trigger: L'Utente seleziona un errore dall'elenco.

UC13.1.1: Visualizzazione del numero della riga

Attore: Utente.

Scenario principale: Il sistema recupera e visualizza il numero della riga

sorgente.

Precondizioni: L'errore è associato a una riga del file.

Postcondizioni: L'Utente può ricondurre l'errore al file originale.

Trigger: Viene aperto il dettaglio di un errore.

UC13.1.2: Visualizzazione del campo interessato

Attore: Utente.

Scenario principale: Il sistema recupera e visualizza il nome del campo sorgente o di destinazione interessato.

Precondizioni: L'errore è associato a uno specifico campo.

Postcondizioni: L'Utente conosce l'attributo da esaminare.

Trigger: Viene aperto il dettaglio di un errore.

UC13.1.3: Visualizzazione del codice di errore

Attore: Utente.

Scenario principale: Il sistema recupera e visualizza il codice associato all'errore.

Precondizioni: Alla violazione è stato assegnato un codice.

Postcondizioni: L'errore può essere classificato e raggruppato in modo univoco.

Trigger: Viene aperto il dettaglio di un errore.

UC13.1.4: Visualizzazione del valore non valido

Attore: Utente.

Scenario principale: Il sistema recupera il valore originale e lo visualizza senza modificarlo.

Precondizioni: L'errore è associato a un valore acquisito dal file.

Postcondizioni: L'Utente può confrontare il valore con la regola violata.

Trigger: Viene aperto il dettaglio di un errore.

UC13.1.5: Visualizzazione del messaggio di errore

Attore: Utente.

Scenario principale: Il sistema recupera il messaggio associato al codice e lo visualizza nel dettaglio.

Precondizioni: L'errore dispone di una descrizione leggibile.

Postcondizioni: L'Utente comprende quale correzione può essere necessaria.

Trigger: Viene aperto il dettaglio di un errore.

UC13.1.6: Visualizzazione dei dati completi del record

Attore: Utente.

Scenario principale: Il sistema recupera i dati della riga e li presenta in forma strutturata, evidenziando il campo coinvolto.

Precondizioni: Il record sorgente è stato conservato nell'area di *staging*^G.

Postcondizioni: L'Utente può valutare l'errore nel contesto dell'intero record.

Trigger: L'Utente richiede il contesto completo dell'errore.

3.3.14 UC14 – Correzione degli errori di validazione

UC14: Correzione degli errori di validazione

Grafico UML:

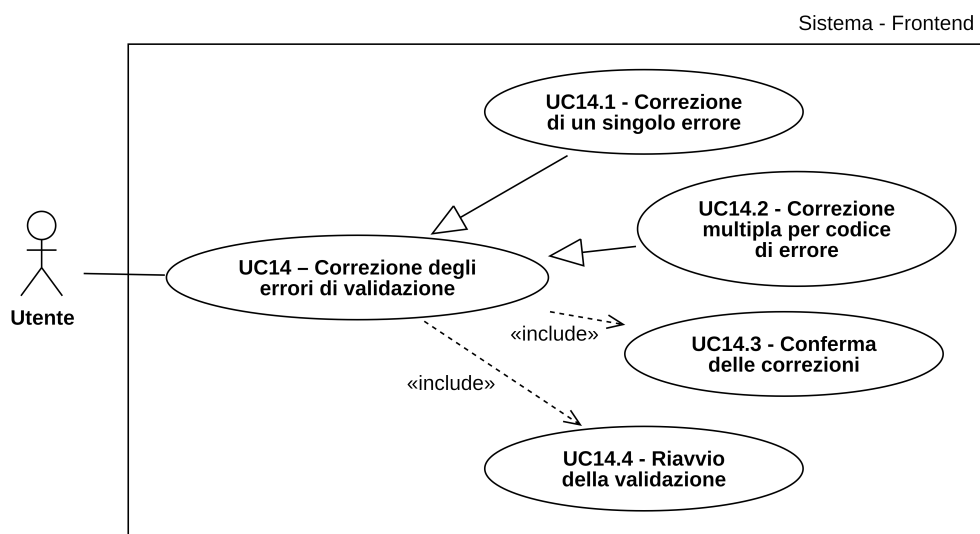


Figura 3.9: UC14: Correzione degli errori di validazione

Attore: Utente.

Scenario principale: L'Utente applica correzioni singole o multiple, le conferma e riavvia la validazione.

Precondizioni: L'esecuzione contiene errori correggibili e non è stata caricata definitivamente.

Postcondizioni: Le correzioni sono salvate e i record interessati dispongono di

un nuovo esito di validazione.

Inclusioni: UC14.3 – Conferma delle correzioni; UC14.4 – Riavvio della validazione.

Specializzazioni: UC14.1 – Correzione di un singolo errore; UC14.2 – Correzione multipla per codice di errore.

Trigger: L'Utente decide di correggere gli errori rilevati in un'esecuzione.

UC14.1: Correzione di un singolo errore

Attore: Utente.

Scenario principale: Il sistema mostra il valore corrente e, quando possibile, propone valori compatibili; l'Utente inserisce o seleziona la correzione; il sistema verifica il formato e prepara la modifica.

Precondizioni: L'errore è correggibile e il record appartiene all'esecuzione corrente.

Postcondizioni: La singola correzione è pronta per essere confermata.

Trigger: L'Utente seleziona il comando di modifica relativo a un errore.

UC14.2: Correzione multipla per codice di errore

Attore: Utente.

Scenario principale: Il sistema mostra i record coinvolti; l'Utente definisce il nuovo valore e lo applica alle righe selezionate.

Precondizioni: Più record presentano lo stesso codice di errore e ammettono una correzione comune.

Postcondizioni: Le correzioni multiple selezionate sono pronte per essere confermate.

Trigger: L'Utente seleziona la correzione multipla per un codice di errore.

UC14.3: Conferma delle correzioni

Attore: Utente.

Scenario principale: L'Utente conferma una correzione tramite il comando di salvataggio o di applicazione; il sistema aggiorna i dati di staging e registra la modifica.

Precondizioni: È presente almeno una modifica valida non ancora confermata.

Postcondizioni: Le correzioni sono salvate e associate all'esecuzione.

Trigger: L'Utente seleziona il comando per salvare o applicare una correzione.

UC14.4: Riavvio della validazione

Attore: Utente.

Scenario principale: Il sistema avvia la validazione, aggiorna lo stato dell'esecuzione, applica le regole e salva i nuovi esiti.

Precondizioni: Le correzioni sono state salvate e l'esecuzione non è in elaborazione.

Postcondizioni: Conteggi ed errori dell'esecuzione riflettono il risultato della nuova validazione.

Trigger: L'Utente richiede una nuova validazione dopo aver confermato le correzioni.

3.3.15 UC15 – Caricamento definitivo dei dati validati

UC15: Caricamento definitivo dei dati validati

Grafico UML:

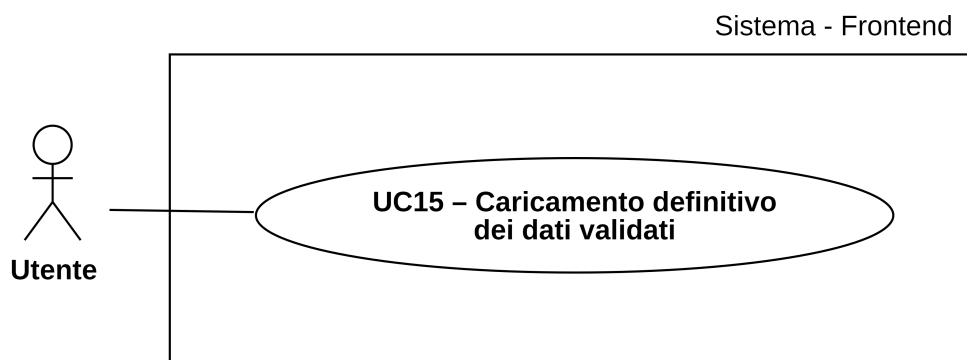


Figura 3.10: UC15: Caricamento definitivo dei dati validati

Attore: Utente.

Scenario principale: L'Utente avvia il caricamento definitivo; il sistema verifica che la validazione dell'intero file sia terminata senza errori, confronta i dati con quelli presenti nel database, inserisce i nuovi record, aggiorna quelli esistenti e registra l'esito del caricamento.

Precondizioni: La validazione dell'intero file CSV è terminata, tutte le righe risultano valide, non sono presenti errori o elaborazioni attive e l'esecuzione non è già stata caricata definitivamente.

Postcondizioni: Tutti i record del file sono inseriti o aggiornati nel database e l'esecuzione conserva il relativo stato finale.

Trigger: L'Utente richiede il caricamento definitivo dopo il completamento con esito positivo della validazione.

3.3.16 UC16 – Logout

UC16: Logout

Grafico UML:

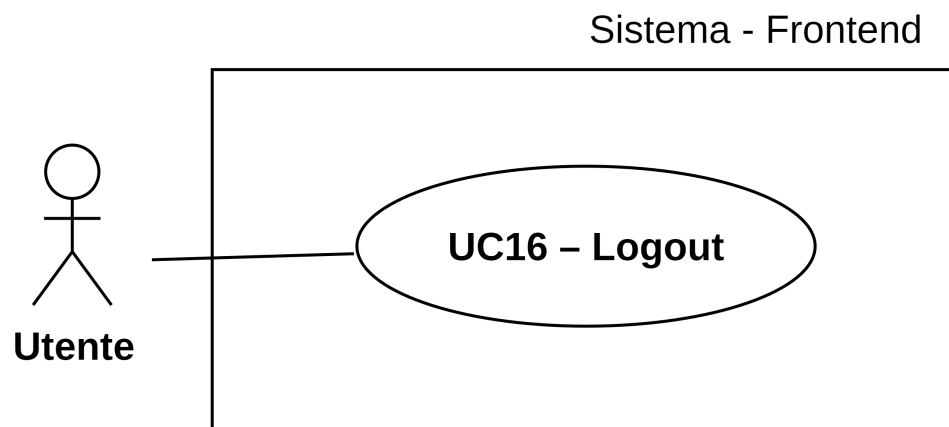


Figura 3.11: UC16: Logout

Attore: Utente.

Scenario principale: Il sistema invalida la sessione e reindirizza l'Utente alla pagina di autenticazione.

Precondizioni: L'Utente dispone di una sessione autenticata.

Postcondizioni: La sessione non può più essere utilizzata per accedere alle risorse protette.

Trigger: L'Utente seleziona il comando di disconnessione.

3.4 Tabella dei requisiti

I requisiti raccolti sono organizzati in tre categorie:

- **Requisiti funzionali (RF):** descrivono le funzionalità che il sistema deve offrire;
- **Requisiti di qualità (RQ):** descrivono gli attributi qualitativi del sistema, quali prestazioni, scalabilità, affidabilità e manutenibilità;

- **Requisiti di vincolo (RV):** descrivono i limiti tecnici e organizzativi entro cui il sistema deve operare.

La priorità di ogni requisito è codificata direttamente nel suo identificativo secondo lo schema seguente:

- **XX-OB-NN – Obbligatorio:** requisito indispensabile per il corretto funzionamento del sistema;
- **XX-DE-NN – Desiderabile:** requisito non critico ma fortemente raccomandato;
- **XX-OP-NN – Opzionale:** requisito facoltativo, implementabile in fasi successive.

Ad esempio, **RF-OB-01** indica il primo requisito funzionale obbligatorio, mentre **RQ-DE-01** indica il primo requisito di qualità desiderabile. La colonna **Fonti** riporta il caso d'uso o il soggetto dal quale deriva ciascun requisito.

3.4.1 Requisiti funzionali

3.4.1.1 Obbligatori

Codice	Descrizione	Fonti
RF-OB-01	L'Utente deve poter autenticarsi nel sistema.	UC01
RF-OB-02	L'Utente deve poter inserire il proprio username nel modulo di autenticazione.	UC01.1
RF-OB-03	L'Utente deve poter inserire la propria password nel modulo di autenticazione.	UC01.2
RF-OB-04	L'Utente deve visualizzare un messaggio di errore qualora le credenziali inserite non siano valide.	UC02
RF-OB-05	L'Utente deve poter accedere alla sezione dedicata al caricamento dei file.	UC03
RF-OB-06	L'Utente deve poter visualizzare le istruzioni per il caricamento del file CSV.	UC03.1
RF-OB-07	L'Utente deve poter visualizzare il collegamento al template CSV.	UC03.2
RF-OB-08	L'Utente deve poter scaricare il template CSV.	UC03.2.1
RF-OB-09	L'Utente deve poter selezionare un file CSV dal proprio dispositivo.	UC03.3
RF-OB-10	L'Utente deve poter caricare il file CSV selezionato.	UC03
RF-OB-11	L'Utente deve visualizzare un messaggio di errore qualora il caricamento del file non vada a buon fine.	UC04
RF-OB-12	Il sistema deve rifiutare i file non conformi al formato CSV previsto.	UC03, UC04
RF-OB-13	L'Utente deve poter avviare manualmente il flusso di integrazione.	UC08
Continua nella prossima pagina...		

Continuazione della tabella

Codice	Descrizione	Fonti
RF-OB-14	Il sistema deve registrare una nuova esecuzione all'avvio del flusso di integrazione.	UC08
RF-OB-15	L'Utente deve poter consultare le esecuzioni associate al proprio tenant.	UC09
RF-OB-16	L'Utente deve poter visualizzare l'elenco delle esecuzioni.	UC09
RF-OB-17	Il sistema deve impedire all'Utente di accedere alle risorse appartenenti ad altri tenant.	UC03–UC15
RF-OB-18	L'Utente deve poter accedere ai risultati di una singola esecuzione.	UC12
RF-OB-19	L'Utente deve poter visualizzare il dettaglio di una singola esecuzione.	UC12
RF-OB-20	L'Utente deve poter visualizzare il nome del file associato all'esecuzione.	UC12.1
RF-OB-21	L'Utente deve poter visualizzare lo stato dell'esecuzione.	UC12.2
RF-OB-22	L'Utente deve poter visualizzare il numero complessivo di righe del file.	UC12.3
RF-OB-23	L'Utente deve poter visualizzare il numero di righe valide.	UC12.4
RF-OB-24	L'Utente deve poter visualizzare il numero di righe non valide.	UC12.5
RF-OB-25	L'Utente deve poter visualizzare la data e l'ora di avvio dell'esecuzione.	UC12.6
RF-OB-26	L'Utente deve poter visualizzare la data e l'ora di conclusione dell'esecuzione oppure l'indicazione che l'esecuzione non è ancora terminata.	UC12.7
RF-OB-27	Il sistema deve validare i dati contenuti nel file CSV.	UC08
RF-OB-28	Il sistema deve distinguere le righe valide dalle righe non valide.	UC08
RF-OB-29	L'Utente deve poter accedere agli errori rilevati durante la validazione.	UC13
RF-OB-30	L'Utente deve poter visualizzare l'elenco degli errori di validazione.	UC13
RF-OB-31	L'Utente deve poter visualizzare il dettaglio di un errore di validazione.	UC13.1
RF-OB-32	L'Utente deve poter visualizzare il numero della riga contenente l'errore.	UC13.1.1
RF-OB-33	L'Utente deve poter visualizzare il campo interessato dall'errore.	UC13.1.2
RF-OB-34	L'Utente deve poter visualizzare il codice associato all'errore.	UC13.1.3
RF-OB-35	L'Utente deve poter visualizzare il valore non valido.	UC13.1.4
RF-OB-36	L'Utente deve poter visualizzare il messaggio associato all'errore.	UC13.1.5
RF-OB-37	L'Utente deve poter visualizzare i dati completi del record contenente l'errore.	UC13.1.6
RF-OB-38	L'Utente deve poter accedere alla funzionalità di correzione degli errori.	UC14
RF-OB-39	L'Utente deve poter correggere un singolo errore di validazione.	UC14.1
RF-OB-40	Il sistema deve salvare la correzione effettuata dall'Utente.	UC14.3
RF-OB-41	L'Utente deve poter confermare le correzioni effettuate.	UC14.3
RF-OB-42	L'Utente deve poter riavviare la validazione dei dati corretti.	UC14.4
RF-OB-43	Il sistema deve validare nuovamente i dati modificati.	UC14.4
Continua nella prossima pagina. . .		

Continuazione della tabella

Codice	Descrizione	Fonti
RF-OB-44	Il sistema deve consentire il caricamento definitivo solamente dopo che la validazione dell'intero file è terminata senza errori.	UC15
RF-OB-45	Il sistema deve individuare i dati non ancora presenti nel database.	UC15
RF-OB-46	Il sistema deve inserire nel database i dati non ancora presenti.	UC15
RF-OB-47	Il sistema deve individuare i dati già presenti nel database.	UC15
RF-OB-48	Il sistema deve aggiornare i dati già presenti nel database.	UC15
RF-OB-49	L'Utente deve poter effettuare il logout dalla piattaforma.	UC16
RF-OB-50	Il sistema deve terminare la sessione dell'Utente in seguito al logout.	UC16

Tabella 3.2: Requisiti funzionali obbligatori

3.4.1.2 Desiderabili

Codice	Descrizione	Fonti
RF-DE-01	Il sistema dovrebbe generare una proposta di mappatura dei campi del file CSV.	UC05
RF-DE-02	L'Utente dovrebbe poter visualizzare la mappatura proposta dal sistema.	UC05.1
RF-DE-03	L'Utente dovrebbe poter visualizzare i campi presenti nel file CSV.	UC05.1.1
RF-DE-04	L'Utente dovrebbe poter visualizzare la corrispondenza proposta per ogni campo del file CSV.	UC05.1.2
RF-DE-05	L'Utente dovrebbe poter visualizzare alcuni valori di esempio associati a ogni campo.	UC05.1.3
RF-DE-06	L'Utente dovrebbe poter riconoscere quali campi risultano mappati.	UC05.1.4
RF-DE-07	L'Utente dovrebbe poter modificare la mappatura di un campo.	UC05.2
RF-DE-08	L'Utente dovrebbe poter confermare la mappatura dei campi.	UC05.3
RF-DE-09	L'Utente dovrebbe poter annullare la mappatura dei campi.	UC05.4
RF-DE-10	L'Utente dovrebbe visualizzare un messaggio di errore in presenza di una colonna duplicata.	UC06
RF-DE-11	L'Utente dovrebbe visualizzare un messaggio di errore in presenza di una colonna obbligatoria mancante.	UC07
RF-DE-12	Il sistema dovrebbe memorizzare una mappatura precedentemente confermata.	UC05.3
RF-DE-13	Il sistema dovrebbe riutilizzare automaticamente una mappatura precedentemente confermata quando risulta compatibile con la struttura del file corrente.	UC05
RF-DE-14	L'Utente dovrebbe poter visualizzare alcuni valori di esempio delle colonne del file insieme alle associazioni proposte.	UC05.1.3
RF-DE-15	L'Utente dovrebbe poter impostare filtri sull'elenco delle esecuzioni.	UC10
RF-DE-16	L'Utente dovrebbe poter filtrare le esecuzioni in base allo stato.	UC10.1
RF-DE-17	L'Utente dovrebbe poter filtrare le esecuzioni in base al nome del file.	UC10.2
Continua nella prossima pagina...		

Continuazione della tabella

Codice	Descrizione	Fonti
RF-DE-18	L'Utente dovrebbe poter filtrare le esecuzioni in base a un intervallo temporale.	UC10.3
RF-DE-19	L'Utente dovrebbe poter correggere contemporaneamente più errori caratterizzati dallo stesso codice.	UC14.2
RF-DE-20	Il sistema dovrebbe predisporre la correzione multipla per tutti gli errori selezionati.	UC14.2
RF-DE-21	Il sistema dovrebbe aggiornare automaticamente lo stato visualizzato durante l'elaborazione.	UC12.2
RF-DE-22	Il sistema dovrebbe proporre valori compatibili durante la correzione di un errore.	UC14.1
RF-DE-23	L'Utente dovrebbe poter visualizzare un riepilogo dell'esecuzione conclusa.	UC12
RF-DE-24	L'Utente dovrebbe poter reimpostare i filtri e ripristinare l'elenco completo delle esecuzioni.	UC11

Tabella 3.3: Requisiti funzionali desiderabili

3.4.1.3 Opzionali

Codice	Descrizione	Fonti
RF-OP-01	L'Utente potrebbe programmare l'avvio automatico di un'integrazione.	UC08 (sviluppo futuro)
RF-OP-02	La piattaforma potrebbe supportare formati di input differenti dal CSV.	UC03 (sviluppo futuro)

Tabella 3.4: Requisiti funzionali opzionali

3.4.2 Requisiti di qualità

3.4.2.1 Obbligatori

Codice	Descrizione	Fonti
RQ-OB-01	Un errore durante la normalizzazione o la validazione non deve modificare i dati presenti nelle tabelle definitive.	UC08
RQ-OB-02	L'elaborazione deve preservare l'integrità delle relazioni tra i dati.	UC08, UC15
RQ-OB-03	Lo stato mostrato deve rappresentare correttamente la condizione corrente dell'esecuzione.	UC12.2
RQ-OB-04	Ogni errore di validazione deve essere associato alla riga che lo ha generato.	UC13.1.1
RQ-OB-05	Ogni errore di validazione deve essere associato al campo che lo ha generato.	UC13.1.2
RQ-OB-06	Il messaggio di errore deve indicare chiaramente la causa dell'anomalia.	UC13.1.5
Continua nella prossima pagina...		

Continuazione della tabella

Codice	Descrizione	Fonti
RQ-OB-07	Le correzioni effettuate non devono modificare record diversi da quelli selezionati.	UC14
RQ-OB-08	La rivalidazione deve utilizzare i dati aggiornati dall'Utente.	UC14.4
RQ-OB-09	Le operazioni eseguite durante il flusso devono essere tracciabili.	Tutor aziendale

Tabella 3.5: Requisiti di qualità obbligatori

3.4.2.2 Desiderabili

Codice	Descrizione	Fonti
RQ-DE-01	La proposta di mappatura dovrebbe distinguere chiaramente i campi mappati da quelli non mappati.	UC05.1
RQ-DE-02	La modifica della mappatura dovrebbe essere possibile senza modificare il file originale.	UC05.2
RQ-DE-03	L'avvio del flusso non dovrebbe bloccare l'utilizzo dell'interfaccia web.	UC08
RQ-DE-04	L'elenco delle esecuzioni dovrebbe essere consultabile anche in presenza di numerosi elementi.	UC09
RQ-DE-05	La visualizzazione degli errori dovrebbe risultare comprensibile anche in presenza di numerose anomalie.	Tutor aziendale
RQ-DE-06	La correzione degli errori dovrebbe richiedere il minor numero possibile di operazioni.	Tutor aziendale
RQ-DE-07	Il sistema dovrebbe completare l'elaborazione di un file contenente 100.000 righe in meno di 10 minuti.	Tutor aziendale

Tabella 3.6: Requisiti di qualità desiderabili

3.4.3 Requisiti di vincolo

3.4.3.1 Obbligatori

Codice	Descrizione	Fonti
RV-OB-01	Il sistema deve poter essere distribuito su <i>Google Cloud Platform (GCP)</i> ^G .	Tutor aziendale
RV-OB-02	Il sistema deve essere integrabile nell'infrastruttura aziendale di Sinesy.	Tutor aziendale
RV-OB-03	Il backend deve essere sviluppato utilizzando <i>Node.js</i> ^G e <i>TypeScript</i> ^G .	Tutor aziendale
RV-OB-04	Il frontend deve essere sviluppato utilizzando <i>React</i> ^G e <i>TypeScript</i> .	Tutor aziendale
RV-OB-05	L'autenticazione degli utenti deve essere gestita mediante <i>Firebase Authentication</i> ^G .	Tutor aziendale
RV-OB-06	Il sistema deve adottare un'architettura multi-tenant.	Tutor aziendale
Continua nella prossima pagina...		

Continuazione della tabella

Codice	Descrizione	Fonti
RV-OB-07	Le risorse associate a ciascun tenant devono essere isolate da quelle degli altri tenant.	Tutor aziendale

Tabella 3.7: Requisiti di vincolo obbligatori

3.4.4 Riepilogo dei requisiti

La tabella seguente riepiloga il numero di requisiti individuati, suddivisi per tipologia e priorità.

Tipologia	Obbligatori	Desiderabili	Opzionali	Totale
Requisiti funzionali	50	24	2	76
Requisiti di qualità	9	7	0	16
Requisiti di vincolo	7	0	0	7
Totale	66	31	2	99

Tabella 3.8: Riepilogo dei requisiti per tipologia e priorità

Capitolo 4

Analisi tecnologica e scelta degli strumenti

Questo capitolo descrive il percorso seguito per scegliere lo strumento di integrazione. Dopo aver presentato l'infrastruttura e i flussi già presenti in FourWayShop, vengono richiamati i concetti teorici utili al confronto. Le sezioni successive illustrano la ricerca iniziale, i Proof of Concept realizzati e la scelta finale.

4.1 Contesto tecnologico

FourWayShop è composta da diversi componenti, sviluppati in momenti differenti e collegati agli stessi sistemi di persistenza. L'infrastruttura è ospitata su Google Cloud Platform e affianca applicazioni *legacy*^G a servizi più recenti basati su microservizi.

Le applicazioni più datate sono eseguite in una macchina virtuale. Comprendono un backend e un frontend sviluppati in Java e distribuiti tramite Apache Tomcat. Altri servizi Java sono ospitati su Google App Engine ed espongono API usate anche dalle applicazioni mobili. Le funzionalità più recenti sono invece sviluppate in Node.js e distribuite tramite Google Cloud Run.

La Figura 4.1 riassume questa struttura e mostra i collegamenti tra i componenti applicativi e i sistemi di persistenza.

I dati sono distribuiti tra tre sistemi, usati per scopi diversi. MySQL contiene le principali anagrafiche e i dati relazionali. Firebase, in modalità Datastore, gestisce vendite, documenti e giacenze, mentre BigQuery viene usato per l'analisi dei dati. La presenza di tecnologie differenti deriva dalla crescita della piattaforma nel tempo.

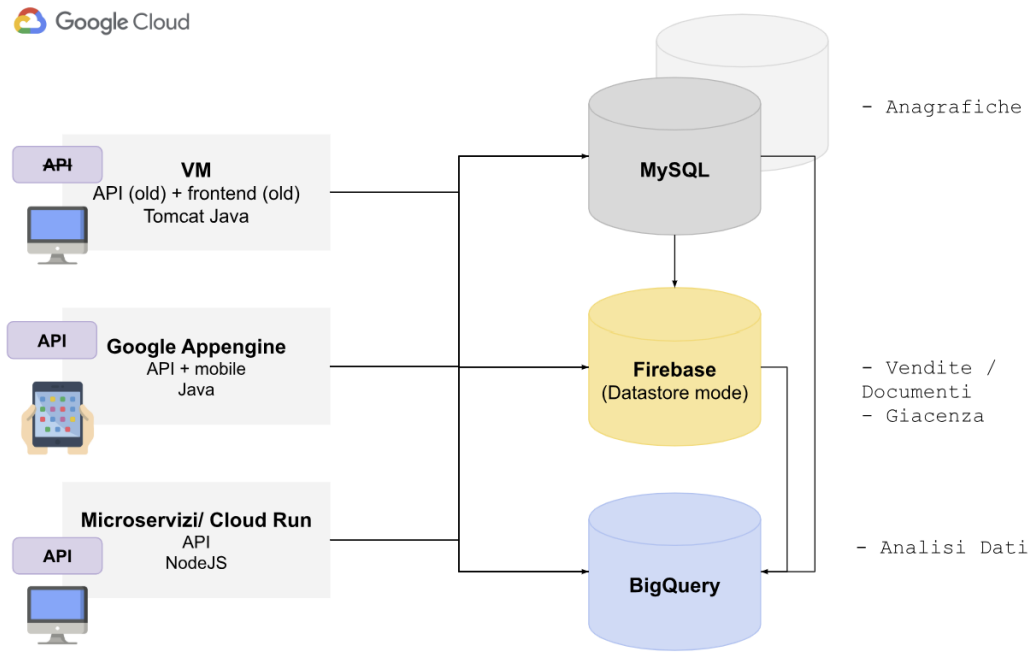


Figura 4.1: Macroarchitettura di FourWayShop

I processi di integrazione permettono di importare in FourWayShop i dati ricevuti dai clienti. Come mostra la Figura 4.2, possono essere automatici o schedulati oppure avviati manualmente dagli operatori.

Nei flussi automatici, il cliente deposita un file CSV in un'area *FTP^G* dedicata. Il sistema di monitoraggio rileva il nuovo file e avvia l'elaborazione subito oppure all'orario stabilito. L'applicazione di integrazione legge il tracciato, associa i campi del cliente a quelli previsti da FourWayShop mediante il mapping e invia i dati tramite le API della piattaforma.

Nei flussi manuali, un operatore inserisce o controlla i dati in un foglio di calcolo di Google Workspace. Dopo la validazione, il foglio viene esportato in formato CSV e caricato attraverso un'applicazione web, dalla quale viene avviata l'integrazione.

Le API *REST^G* di FourWayShop coprono due ambiti principali. Il primo riguarda prodotti, clienti e dati associati; il secondo comprende documenti come ordini, ricevute e fatture. Le API consentono di consultare, inserire, aggiornare o eliminare singole risorse e, in alcuni casi, di eseguire operazioni massive [1]. L'applicazione di integrazione fa quindi da collegamento tra i file ricevuti, le regole di mappatura e i servizi di FourWayShop.

Per questo motivo, lo strumento scelto deve sostituire e migliorare l'attuale processo di integrazione, riuscendo a gestire tecnologie e modalità di acquisizione diverse. L'intervento riguarda il flusso con cui i dati vengono acquisiti, trasformati e inviati ai database, mentre i sistemi di persistenza esistenti rimangono

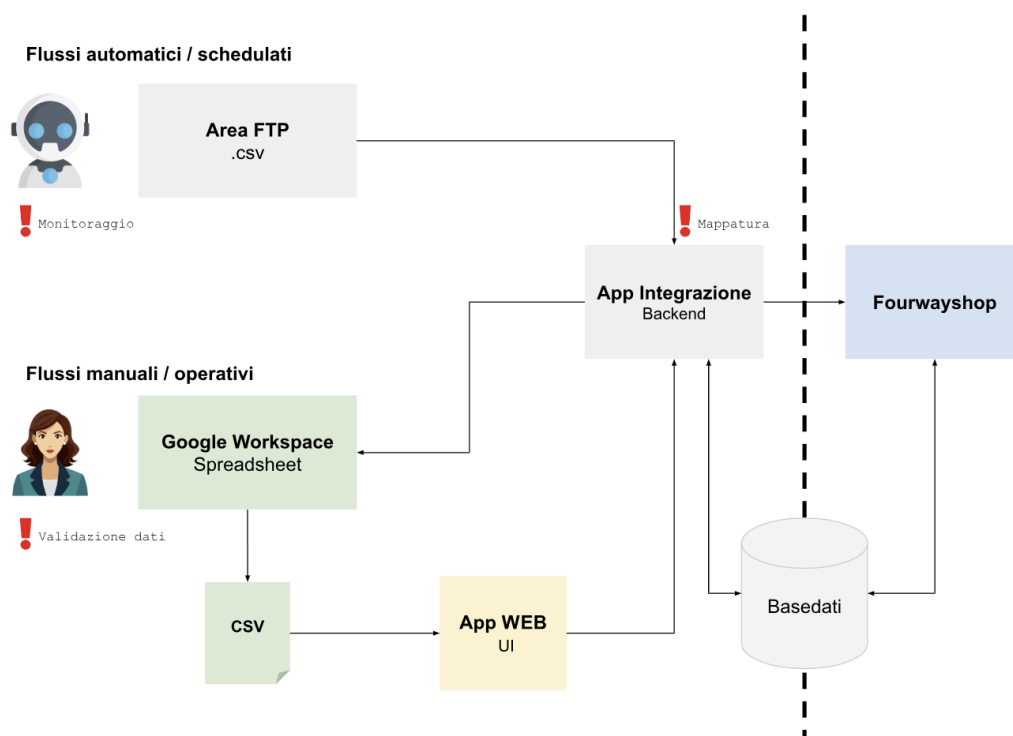


Figura 4.2: Flussi di integrazione esistenti di FourWayShop

invariati. I requisiti funzionali e di qualità sono descritti nel Capitolo 3; le sezioni successive si concentrano invece sulla scelta dello strumento.

4.2 Fondamenti dell'integrazione dei dati

Prima di confrontare gli strumenti è utile richiamare la differenza tra ETL ed *ELT^G* e chiarire il ruolo dell'orchestrazione. In questa sezione vengono descritti solo gli aspetti necessari all'analisi; la loro applicazione verrà approfondita nel capitolo dedicato alla progettazione.

4.2.1 Processi ETL ed ELT

Un processo ETL è composto da tre fasi: i dati vengono estratti dalla sorgente, trasformati e infine caricati nel sistema di destinazione. Nel modello ELT cambia l'ordine delle ultime due fasi: i dati vengono prima caricati e poi trasformati direttamente nel sistema di destinazione.

Il processo previsto per FourWayShop segue soprattutto il modello ETL, perché i file devono essere mappati, normalizzati e validati prima dell'invio alle API della piattaforma.

4.2.2 Flusso di integrazione e orchestrazione

Orchestrare un flusso significa coordinare le attività di acquisizione, trasformazione, validazione e caricamento. Lo strumento deve stabilire l'ordine delle operazioni e gestire gli eventuali errori. Deve inoltre permettere di avviare il flusso manualmente, al verificarsi di un evento oppure secondo una pianificazione. Nel confronto sono quindi importanti sia l'elaborazione dei dati sia il controllo delle esecuzioni.

4.3 Selezione preliminare degli strumenti

Prima di iniziare i test è stata svolta una ricerca sugli strumenti disponibili. Ne sono stati individuati sedici, divisi tra piattaforme ETL visuali, orchestratori di *workflow*^G, *framework*^G di elaborazione distribuita e strumenti per l'ingestione dei dati. Questa prima fase è servita a capire quali soluzioni fossero più vicine alle esigenze del progetto.

In questa fase gli strumenti non sono stati provati direttamente. Le informazioni sono state raccolte dalla documentazione ufficiale, dai siti dei fornitori e da altre risorse tecniche pubbliche [2–18]. Per ogni soluzione sono stati valutati i connettori disponibili, le funzioni di trasformazione e orchestrazione, gli aspetti infrastrutturali e il modello di costo. I giudizi riportati si riferiscono quindi alle esigenze del progetto e non alla qualità generale degli strumenti.

Per rendere il confronto leggibile, i sedici strumenti sono stati divisi in due tabelle. Entrambe usano gli stessi requisiti e la stessa scala di valutazione.

Legenda:

- Soddisfatto
- 🟡 Parziale / configurabile
- ✖ Lacuna significativa

La valutazione considera principalmente le funzionalità offerte direttamente dagli strumenti. Una lacuna non indica necessariamente l'impossibilità di realizzare una determinata operazione, ma può segnalare la necessità di sviluppare codice personalizzato o integrare componenti esterni. I Proof of Concept hanno permesso di verificare concretamente anche questo aspetto.

Requisito	n8n	Airflow	Apache NiFi	Mage.ai	Qlik	GCP Dataflow	Apache Spark	GCP Data Fusion
DATI & CONNETTORI								
CSV / <i>XLSX</i> ^G / file posizionali	●	🟡	●	●	●	🟡	●	●
MySQL / BigQuery	●	🟡	●	●	●	●	●	●
FTP import / export	●	🟡	●	🟡	●	🟡	🟡	●
API REST + <i>SOAP</i> ^G	●	🟡	●	🟡	●	✖	🟡	●
ELABORAZIONE & PERFORMANCE								
Volumi fino a 100.000 righe	🟡	🟡	●	●	●	●	●	●
Trasformazioni dati (ETL)	🟡	✖	●	🟡	●	●	●	●

CAPITOLO 4. ANALISI TECNOLOGICA E SCELTA DEGLI STRUMENTI

Requisito	n8n	Airflow	Apache NiFi	Mage.ai	Qlik	GCP Dataflow	Apache Spark	GCP Data Fusion
ORCHESTRAZIONE & OPERATIVITÀ								
Schedulazione avanzata	🟡	●	●	●	●	✗	✗	●
Monitoraggio e log	🟡	●	●	🟡	●	●	🟡	●
Interfaccia visuale	●	✗	●	🟡	●	✗	✗	●
INFRASTRUTTURA & GOVERNANCE								
Deploy su GCP	🟡	●	🟡	🟡	🟡	●	🟡	●
Scalabilità	🟡	●	🟡	🟡	●	●	●	●
Multi-tenant	🟡	🟡	🟡	✗	●	🟡	🟡	🟡
IAM ^G / gestione ruoli	✗	●	🟡	✗	●	●	🟡	●
COSTO								
Costo licenza	Source available	Open source	Open source	Ibrido	Enterprise	Pay per use	Open source	Pay per use

Tabella 4.1: Prima parte del confronto preliminare degli strumenti

Requisito	GCP Composer	Cloud Run + Scheduler	GCP Workflows	Beam + Dataflow	Dataplex DQ	Prefect	Dagster	Airbyte / Meltano
DATI & CONNETTORI								
CSV / XLSX / file posizionali	🟡	●	✗	🟡	✗	🟡	🟡	●
MySQL / BigQuery	🟡	●	✗	●	●	🟡	🟡	●
FTP import / export	🟡	🟡	✗	🟡	✗	🟡	🟡	🟡
API REST + SOAP	🟡	●	●	🟡	✗	🟡	🟡	●
ELABORAZIONE & PERFORMANCE								
Volumi fino a 100.000 righe	🟡	●	✗	●	✗	🟡	🟡	🟡
Trasformazioni dati (ETL)	✗	●	✗	●	✗	🟡	🟡	🟡
ORCHESTRAZIONE & OPERATIVITÀ								
Schedulazione avanzata	●	🟡	🟡	✗	✗	●	●	🟡
Monitoraggio e log	●	🟡	🟡	●	●	●	●	🟡
Interfaccia visuale	✗	✗	✗	✗	✗	🟡	🟡	🟡
INFRASTRUTTURA & GOVERNANCE								
Deploy su GCP	●	●	●	●	●	🟡	🟡	🟡
Scalabilità	●	●	●	●	●	🟡	🟡	🟡
Multi-tenant	🟡	🟡	🟡	🟡	🟡	🟡	🟡	🟡
IAM / gestione ruoli	●	●	●	●	●	🟡	🟡	🟡
COSTO								
Costo licenza	Pay per use	Pay per use	Pay per use	Pay per use	Pay per use	Ibrido	Ibrido	Open source

Tabella 4.2: Seconda parte del confronto preliminare degli strumenti

I risultati delle Tabelle 4.1 e 4.2 sono stati poi presentati a Sinesy. La scelta degli strumenti da approfondire non si è basata solo sul numero delle lacune,

ma anche sulla loro importanza, sulla versatilità della soluzione e sulla vicinanza alle tecnologie già usate in azienda.

Alcuni strumenti sono stati esclusi perché adatti alla sola elaborazione dei dati e non alla gestione dell'intero flusso. Quando più alternative offrivano funzioni simili, sono state preferite quelle più compatibili con lo stack di Sinesy.

Nessuna soluzione è risultata completa sotto ogni aspetto. La ricerca ha però permesso di restringere il confronto a sei strumenti: n8n, Apache Airflow, Apache NiFi, Google Cloud Composer, Google Cloud Data Fusion e Qlik. Per valutarli in modo più concreto si è deciso di realizzare con ciascuno lo stesso Proof of Concept.

4.4 Scenario e criteri del Proof of Concept

Per confrontare i sei strumenti è stato preparato un semplice processo di importazione dei prodotti. Lo stesso scenario è stato riprodotto con ogni soluzione, così da valutare sia le funzioni disponibili sia il lavoro richiesto per costruire il flusso.

In una cartella accessibile tramite *SSH File Transfer Protocol (SFTP)*^G venivano inseriti due file: uno con le anagrafiche dei prodotti e uno con i relativi codici a barre. Il flusso doveva eseguire questi passaggi:

1. rilevare la presenza dei due file nella cartella SFTP e scaricarli;
2. leggere e normalizzare i dati contenuti nei due tracciati;
3. unire le informazioni, associando a ogni prodotto il relativo codice a barre;
4. interrogare un'API utilizzando il codice del prodotto, per verificare se fosse già presente nei sistemi di FourWayShop;
5. stabilire, in base alla risposta, se fosse necessario inserire oppure aggiornare il prodotto.

Il PoC non modificava realmente il database. Per ogni prodotto produceva soltanto un messaggio con l'operazione da eseguire. In questo modo è stato possibile provare la logica di inserimento e aggiornamento senza rendere la sperimentazione più complessa del necessario.

Durante i test sono stati osservati il supporto a SFTP e alle chiamate API, la capacità di elaborare e unire i file, la gestione delle condizioni, la chiarezza dei log e i tempi di esecuzione. È stata considerata anche la facilità con cui il flusso poteva essere configurato e modificato.

4.5 Valutazione comparativa delle soluzioni

Di seguito vengono riportati i risultati ottenuti con ciascuno strumento. Per ogni soluzione sono presenti una breve descrizione, gli aspetti positivi e negativi emersi durante il PoC e una conclusione.

4.5.1 n8n

Descrizione

n8n è una piattaforma per l'automazione dei workflow, disponibile come servizio cloud oppure in modalità *self-hosted*^G. I flussi vengono costruiti con un editor visuale collegando nodi che rappresentano eventi, operazioni o sistemi esterni. Oltre ai connettori già disponibili, è possibile eseguire richieste *HTTP*^G e aggiungere codice personalizzato [2].

Aspetti positivi

L'editor è risultato semplice da usare e ha permesso di costruire rapidamente il PoC. Per ogni nodo è possibile vedere i dati in ingresso e in uscita, seguendo con facilità le trasformazioni eseguite. I nodi necessari per SFTP, API, condizioni e file erano già presenti. Dove serviva maggiore libertà è stato usato il nodo Code. Anche i log e i messaggi di errore sono risultati chiari, mentre i tempi di esecuzione sono stati contenuti.

Aspetti negativi

Il limite principale ha riguardato la configurazione gratuita usata durante la prova. Questa non permetteva a più colleghi di lavorare facilmente sullo stesso workflow e non offriva una separazione completa degli spazi di lavoro. È stato comunque possibile condividere flussi e configurazioni tramite esportazione e importazione, anche se in modo meno comodo rispetto alle versioni a pagamento.

Conclusione

Il PoC è stato completato senza particolari difficoltà. L'unione tra editor visuale, codice personalizzato e strumenti di controllo ha reso n8n uno dei candidati più adatti.

La Figura 4.3 mostra il workflow realizzato, dall'acquisizione dei due file alla scelta dell'operazione da simulare.

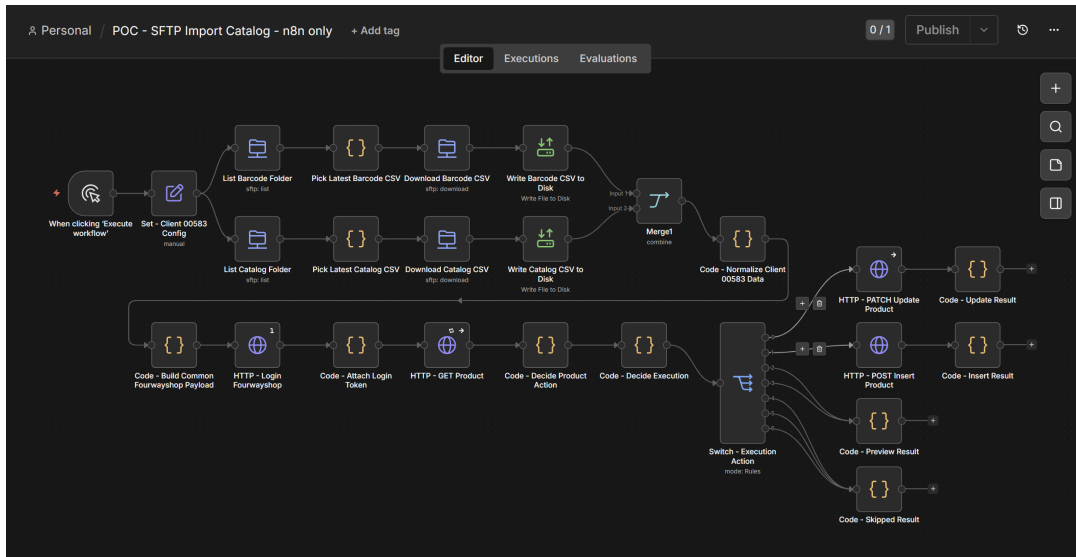


Figura 4.3: Proof of Concept realizzato con n8n

4.5.2 Apache Airflow

Descrizione

Apache Airflow è una piattaforma open source per creare, pianificare e monitorare workflow batch. I flussi sono definiti in Python come DAG^G , composti da task e dipendenze. Airflow coordina le attività, mentre l'elaborazione è affidata al codice Python o a sistemi esterni [3].

Aspetti positivi

La scrittura in Python ha permesso di realizzare il PoC senza particolari vincoli. Il DAG ha reso chiare le dipendenze e lo stato dei task, mentre il codice poteva essere versionato insieme al progetto.

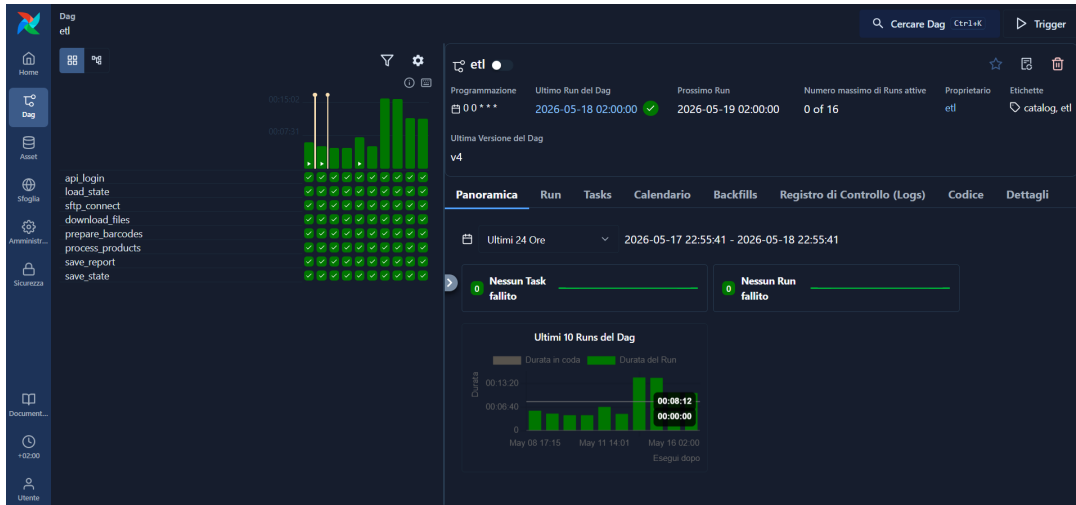
Aspetti negativi

L'interfaccia visuale consente di controllare il workflow, ma non di costruirlo. Acquisizione, trasformazione, chiamate API e produzione del risultato sono state quindi implementate in Python. Airflow è flessibile, ma meno immediato per un gruppo abituato soprattutto a Node.js. Per seguire l'avanzamento dei task è inoltre necessario inserire nel codice log dettagliati.

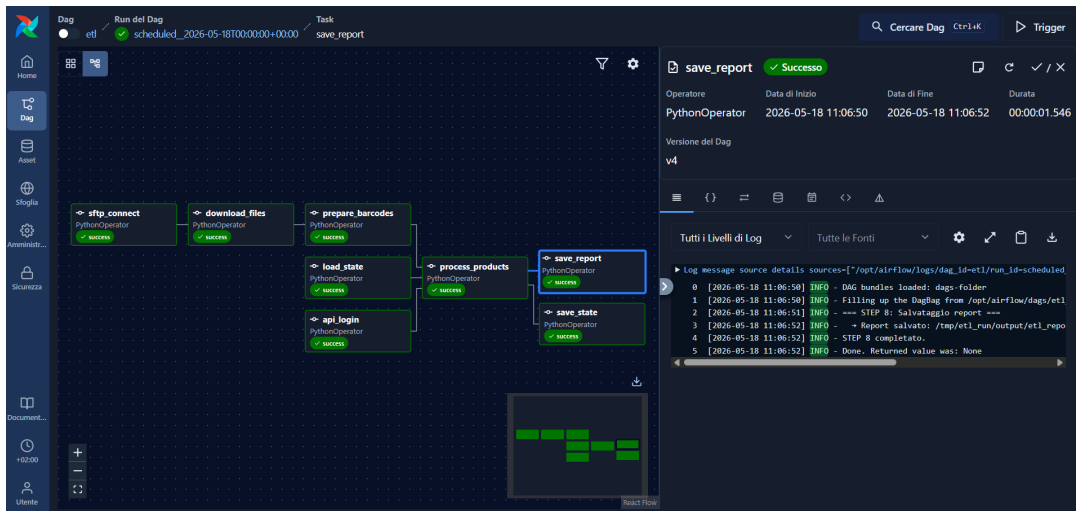
Conclusione

Il PoC è stato completato e Airflow ha permesso di personalizzare ogni parte del flusso. Rispetto a n8n ha però richiesto più codice e più attenzione nella gestione dei log, senza offrire la stessa immediatezza visuale.

La Figura 4.4 mostra il DAG, le esecuzioni e i log dei task.



(a) Panoramica delle esecuzioni del DAG



(b) Struttura del DAG e log di un task

Figura 4.4: Proof of Concept realizzato con Apache Airflow

4.5.3 Apache NiFi

Descrizione

Apache NiFi è una piattaforma open source per la gestione dei flussi di dati. Il workflow viene costruito collegando processori visuali, ognuno dei quali svolge un'operazione di acquisizione, trasformazione, instradamento o scrittura. NiFi consente inoltre di accodare i dati, regolarne il flusso e tracciarne la provenienza [4].

Aspetti positivi

L'interfaccia mostra il percorso dei dati e lo stato dei collegamenti tra i processori. I log sono risultati dettagliati e hanno permesso di capire fino a quale punto fosse arrivata l'elaborazione. I tempi di esecuzione si sono mantenuti nella media rispetto agli altri strumenti.

Aspetti negativi

La configurazione è risultata più complessa rispetto a n8n. Ogni processore richiede di impostare diversi parametri e di conoscere bene il formato dei dati in ingresso e le possibili uscite. Questo ha aumentato il tempo necessario per costruire il PoC. Nell'ambiente provato mancava inoltre un componente immediato, simile al nodo Code di n8n, con cui aggiungere liberamente logica personalizzata.

Conclusione

NiFi ha offerto buoni strumenti di tracciamento, ma il workflow è risultato meno immediato da configurare e personalizzare. Per questo motivo non è stato incluso tra i candidati finali.

La Figura 4.5 mostra una parte del flusso costruito mediante i processori di NiFi.

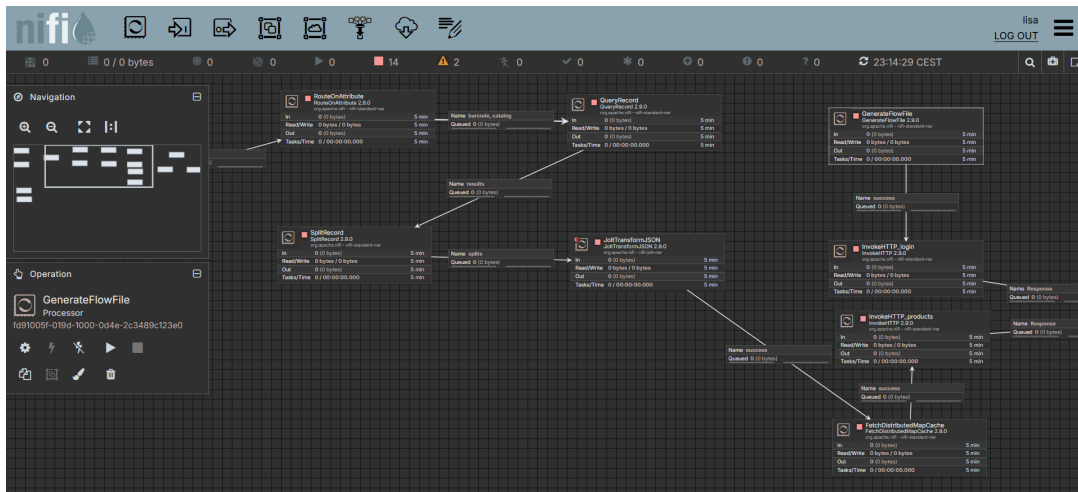


Figura 4.5: Proof of Concept realizzato con Apache NiFi

4.5.4 Google Cloud Composer

Descrizione

Google Cloud Composer è un servizio gestito di Google Cloud basato su Apache Airflow. I workflow continuano a essere scritti in Python, ma non è necessario installare e mantenere direttamente l'orchestratore. Google Cloud gestisce l'ambiente e l'integrazione con i servizi di registrazione dei log, monitoraggio e controllo degli accessi [10].

Aspetti positivi

Composer ha permesso di riutilizzare il modello e la logica sviluppati con Airflow, senza dover preparare e amministrare l'ambiente di esecuzione. Il vantaggio principale è quindi la gestione dell'infrastruttura da parte di Google Cloud.

Aspetti negativi

I limiti riscontrati con Airflow rimangono: il workflow deve essere scritto in Python e i log devono essere aggiunti con cura nel codice. A questo si aggiunge il costo dell'ambiente cloud, risultato elevato rispetto alle dimensioni del PoC e al numero iniziale di flussi.

Conclusione

Dal punto di vista del workflow, Composer si è comportato come Airflow. È rimasto tra i tre candidati finali, ma il vantaggio del servizio gestito non ha compensato il costo dell'ambiente e la minore familiarità di Sinesy con Python. Non viene riportata un'altra figura, perché il DAG e l'interfaccia sono gli stessi mostrati nella Figura 4.4.

4.5.5 Google Cloud Data Fusion

Descrizione

Google Cloud Data Fusion è un servizio gestito per la creazione di pipeline di integrazione. L'interfaccia Studio offre un editor visuale e diversi plugin per collegare sorgenti e destinazioni e applicare le trasformazioni necessarie. Il servizio è basato sul progetto open source *Cask Data Application Platform (CDAP)*^G e non richiede la gestione diretta dell'infrastruttura [9].

Aspetti positivi

L'editor permette di rappresentare visivamente la pipeline e le trasformazioni applicate. Essendo un servizio Google Cloud, riduce inoltre le attività necessarie per gestire l'infrastruttura.

Aspetti negativi

Durante la prova non è stato possibile trovare o configurare alcuni dei componenti necessari, soprattutto quelli richiesti per completare la logica successiva all'unione dei dati. Già nelle prime fasi il lavoro richiesto è risultato troppo elevato rispetto al risultato ottenuto. Anche il costo dell'istanza è apparso alto per una prova di dimensioni contenute.

Conclusione

Il PoC non è stato completato. La mancanza dei componenti necessari, insieme al costo e al tempo richiesto, ha portato a escludere Data Fusion senza proseguire la prova.

La Figura 4.6 mostra la pipeline parziale realizzata prima dell'interruzione della prova.

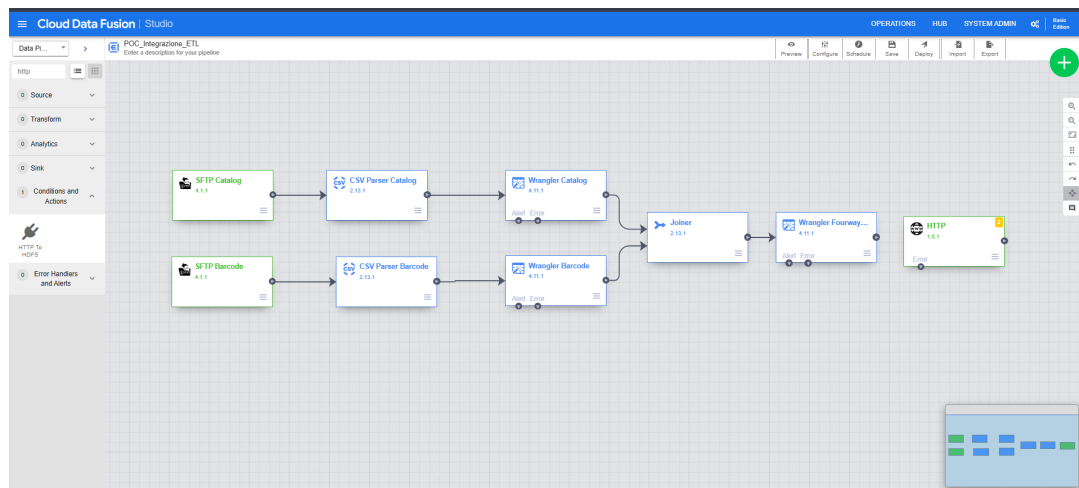


Figura 4.6: Pipeline parziale realizzata con Google Cloud Data Fusion

4.5.6 Qlik

Descrizione

Per il PoC è stato usato Qlik Automate, il modulo no-code di Qlik Cloud per l'automazione dei processi [6]. Il workflow viene costruito mettendo in sequenza blocchi che rappresentano operazioni, condizioni o collegamenti con servizi

esterni. Qlik è stato incluso nella prova anche perché Sinesy disponeva già di una licenza.

Aspetti positivi

Qlik Automate offre blocchi visuali e un'infrastruttura già pronta per esecuzione, controllo e sicurezza. Può quindi essere utile per creare workflow semplici senza scrivere codice. La licenza aziendale ha inoltre permesso di svolgere il test senza nuovi acquisti.

Aspetti negativi

La configurazione è risultata meno pratica rispetto a n8n. La disposizione dei blocchi è più vincolata e rende meno leggibili i flussi lunghi. Anche la personalizzazione ha richiesto più tempo, perché era necessario costruire nel dettaglio le operazioni più semplici usando diversi blocchi di base. Per questo motivo, Qlik è risultato meno flessibile rispetto a uno strumento che permette di combinare nodi visuali e codice.

Conclusione

Qlik Automate può essere adatto ad automazioni no-code non troppo complesse, soprattutto quando si vuole usare un'infrastruttura enterprise già gestita. Per Sinesy, però, i vincoli dell'editor e la scarsa flessibilità lo hanno reso poco adatto ai workflow previsti. Per questo motivo è stato escluso.

La Figura 4.7 mostra l'automazione costruita durante la sperimentazione.

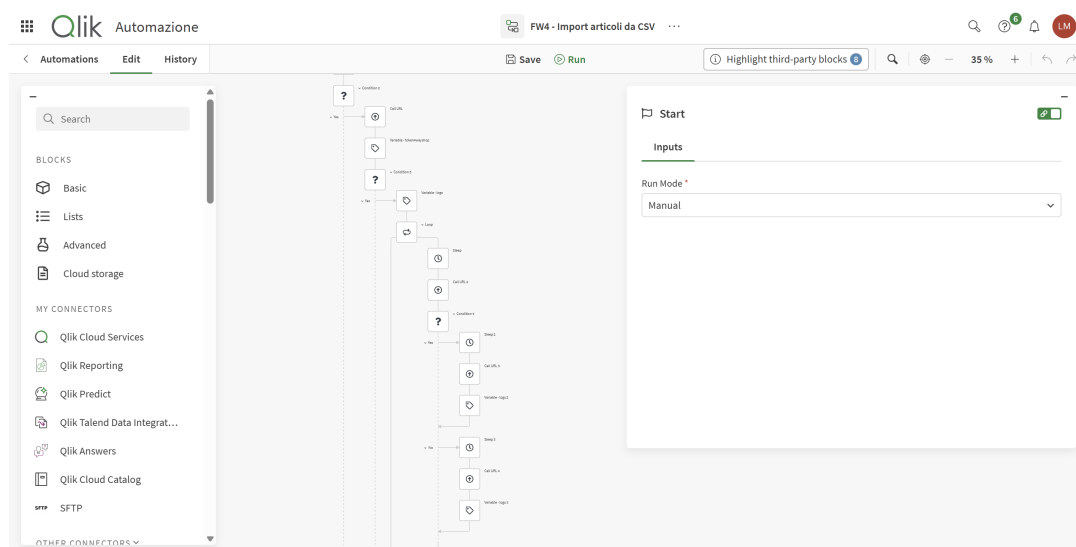


Figura 4.7: Proof of Concept realizzato con Qlik

4.5.7 Esito della sperimentazione

Al termine dei test sono stati esclusi Apache NiFi, Google Cloud Data Fusion e Qlik. NiFi e Qlik sono risultati meno pratici da configurare e poco flessibili, mentre Data Fusion non ha permesso di completare il PoC con i componenti disponibili.

Il confronto finale si è concentrato su n8n, Apache Airflow e Google Cloud Composer. La scelta è ricaduta su n8n, che ha permesso di completare il workflow con meno lavoro rispetto agli altri due. L'editor visuale, la possibilità di aggiungere codice, i connettori già disponibili e la consultazione dei log hanno reso lo strumento più adatto al progetto.

Rimane il limite della collaborazione tra più utenti, non disponibile nella configurazione gratuita usata per il PoC. Durante il progetto ogni workflow è stato condiviso esportando e importando il relativo file *JSON^G*, che contiene la struttura e la configurazione del flusso. La procedura è meno immediata rispetto a quella offerta dalla versione a pagamento, ma è stata sufficiente per proseguire il lavoro.

Capitolo 5

Progettazione della soluzione

Questo capitolo descrive la progettazione del sistema ETL, presentandone l'architettura, le tecnologie, il modello dati e il flusso di importazione orchestrato tramite n8n.

5.1 Criteri progettuali

A partire dai requisiti e dai vincoli presentati nei Capitoli 3 e 4, la progettazione segue tre criteri principali: controllare e preparare i dati prima di caricarli nelle tabelle definitive, conservare lo stato di ogni importazione e assegnare a ciascun componente responsabilità precise.

5.2 Architettura generale del sistema

Dal confronto svolto nel capitolo precedente, n8n è risultato lo strumento più adatto per orchestrare il flusso ETL. Tuttavia, come le altre soluzioni analizzate, non era sufficiente da solo a soddisfare tutti i requisiti del progetto. È stato quindi affiancato da un frontend, che fornisce l'interfaccia grafica, e da un backend, che mette in comunicazione i diversi componenti e gestisce la logica applicativa.

Il sistema ETL è quindi composto da più parti, ciascuna dedicata a un compito preciso. L'architettura può essere suddivisa in cinque macroaree, riportate nella Tabella 5.1.

Macroarea	Componenti principali	Responsabilità
Presentazione	Frontend web	Fornisce all'Utente le pagine per avviare e controllare le importazioni, consultare gli errori e inserire le correzioni.
Applicazione	Backend applicativo	Gestisce l'accesso al sistema, espone le API, applica le regole applicative e controlla le operazioni sui dati.
Orchestrazione	n8n	Esegue e coordina le attività ETL definite nei workflow.
Persistenza	keepit_svil_bulk e keepit_svil in MySQL	Mantiene separati i dati elaborati dall'ETL da quelli definitivi utilizzati da FourWayShop.
Servizi di supporto	Firebase Authentication, Redis e PostgreSQL	Supporta l'autenticazione, la gestione temporanea dei permessi e il funzionamento interno di n8n.

Tabella 5.1: Macroaree dell'architettura del sistema ETL

La Figura 5.1 mostra come queste aree sono collegate. Il diagramma rappresenta la struttura del sistema e le comunicazioni consentite tra i componenti, non l'ordine delle fasi dell'importazione, descritto nella Sezione 5.7.

Il frontend costituisce il punto di contatto con l'Utente, ma non accede direttamente a n8n o ai database. Tutte le richieste passano attraverso il backend, che rappresenta il punto centrale del sistema e verifica che ogni operazione sia autorizzata e riferita all'azienda corretta.

Il backend contiene le API e le regole applicative, mentre n8n si occupa dell'orchestrazione. Questa distinzione evita di trasferire nei workflow la logica legata agli utenti, ai tenant e al dominio di FourWayShop. I due componenti comunicano attraverso *webhook*^G e servizi HTTP: n8n coordina le fasi del processo, mentre il backend stabilisce quali operazioni possono essere eseguite.

Anche la persistenza è suddivisa. Il database MySQL `keepit_svil_bulk` contiene staging, tabelle di frontiera, run, errori e informazioni di audit; `keepit_svil` contiene invece i dati definitivi di FourWayShop. PostgreSQL è riservato alle informazioni interne di n8n. Firebase Authentication e Redis forniscono servizi trasversali e non partecipano alla conservazione dei dati importati.

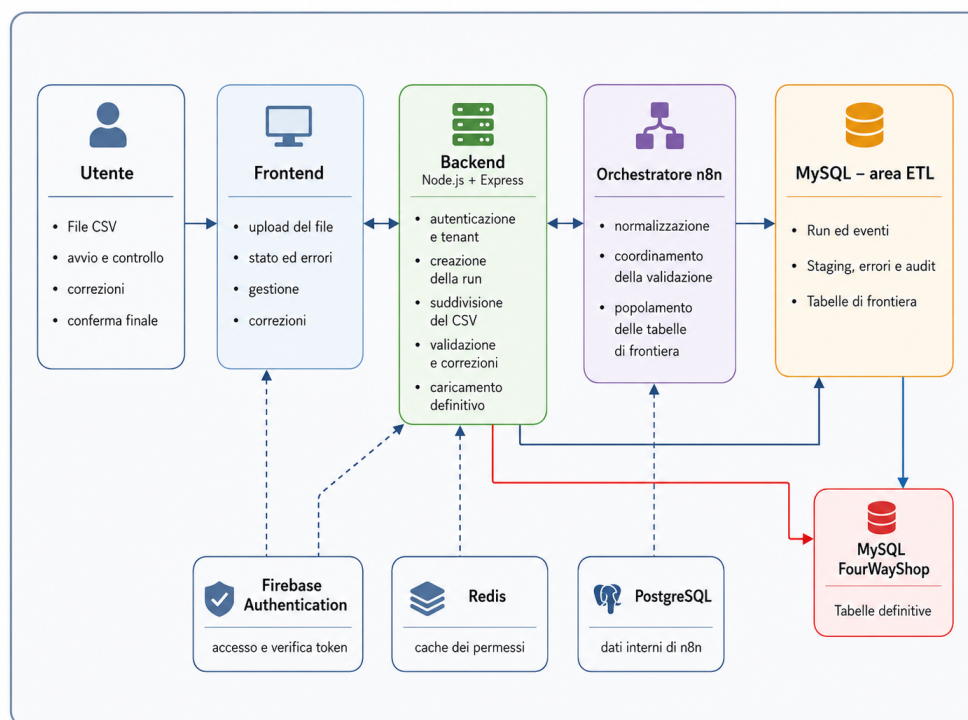


Figura 5.1: Architettura generale del sistema ETL

5.3 Tecnologie utilizzate

Le tabelle seguenti riportano le tecnologie utilizzate, le relative versioni e il loro ruolo nel progetto. Il simbolo “–” indica che non è stata definita una versione precisa. Per n8n è riportato il tag **stable**, che non corrisponde a una versione specifica.

5.3.1 Backend

Tecnologia	Versione	Ruolo nel progetto
Node.js	22.14.0	Ambiente nel quale viene eseguito il backend.
TypeScript	5.9.3	Linguaggio usato per sviluppare il backend. Il controllo dei tipi aiuta a individuare gli errori prima dell'esecuzione.
Express	4.22.2	Framework che riceve le richieste HTTP e organizza le API del backend.
Multer	2.1.1	Permette al backend di ricevere i file CSV caricati dall'Utente.

Tecnologia	Versione	Ruolo nel progetto
Axios	1.16.1	Permette al backend di comunicare con n8n attraverso i webhook.
Zod	3.25.76	Controlla la struttura dei dati ricevuti dalle API e definisce gli schemi usati per descrivere le risposte.
OpenAPI e Swagger UI	3.0.0 / 5.0.1	Documentano le API e permettono di provarle attraverso un'interfaccia web.
Sequelize e MySQL2	6.37.8 / 3.22.3	Permettono al backend di collegarsi a MySQL e di leggere o modificare i dati.
Firebase Admin	13.10.0	Verifica i token di accesso e gestisce le operazioni amministrative sugli account.
Pino	9.14.0	Registra in modo strutturato le attività e gli errori del backend.
Helmet	8.1.0	Aggiunge alle risposte HTTP alcune impostazioni di sicurezza.
<i>CORS</i> ^G	2.8.6	Configura le origini dalle quali il browser consente di richiamare le API.
Express Rate Limit	7.5.1	Limita il numero di richieste ricevute in un determinato intervallo di tempo.

Tabella 5.2: Tecnologie utilizzate nel backend

TypeScript controlla i tipi durante lo sviluppo, ma non può garantire che i dati ricevuti durante l'esecuzione siano corretti. Zod controlla quindi le richieste effettive. Gli stessi schemi vengono usati per descrivere le risposte nella documentazione OpenAPI, mantenendola coerente con il codice.

Sequelize permette di accedere al database tramite modelli TypeScript. Per le operazioni ETL più complesse è possibile eseguire direttamente query *SQL*^G. Più modifiche possono inoltre essere riunite in una transazione.

5.3.2 Frontend

Tecnologia	Versione	Ruolo nel progetto
React	19.2.6	Libreria usata per costruire l'interfaccia attraverso componenti riutilizzabili.
TypeScript	5.9.3	Linguaggio usato per definire con precisione componenti, modelli e dati scambiati con il backend.
Vite	5.4.2	Avvia il frontend durante lo sviluppo e prepara i file da distribuire.
PrimeReact	10.9.6	Fornisce componenti grafici già pronti, come tabelle, pulsanti, finestre di dialogo e campi di input.
Tailwind <i>CSS</i> ^G	4.3.0	Aiuta a definire la disposizione, le spaziature e lo stile degli elementi della pagina.
Phosphor Icons	2.1.10	Fornisce le icone utilizzate nell'interfaccia.
TanStack Router	1.170.5	Collega gli indirizzi dell'applicazione alle relative pagine e gestisce la navigazione.
TanStack Query	5.100.11	Recupera i dati dal backend, li conserva temporaneamente e li aggiorna quando necessario.
Luxon	3.7.2	Gestisce date, orari e conversioni di fuso orario nell'interfaccia.
Axios	1.16.1	Permette al frontend di comunicare con le API del backend.
Zod	3.25.76	Controlla che i dati restituiti dal backend abbiano la struttura attesa.
React Hook Form	7.76.0	Gestisce i valori inseriti dall'Utente e gli errori presenti nei moduli.
i18next	25.10.10	Mantiene i testi tradotti separati dal codice e permette di cambiare lingua.
Firebase SDK	12.13.0	Gestisce l'accesso dell'Utente, la sessione e il token inviato al backend.

Tabella 5.3: Tecnologie utilizzate nel frontend

TanStack Query è particolarmente utile nelle pagine che mostrano l'avanzamento delle importazioni. Aggiorna periodicamente i dati della run e richiede nuove informazioni dopo una correzione, una rivalidazione o un caricamento.

L'interfaccia mostra così uno stato sempre allineato con quello conservato dal backend.

Anche nel frontend Zod completa i controlli di TypeScript. Prima di usare una risposta del backend, verifica che i dati ricevuti corrispondano al modello previsto dall'interfaccia.

5.3.3 Database

Tecnologia	Versione	Ruolo nel progetto
MySQL	–	Ospita <code>keepit_svil</code> , il database finale di FourWayShop, e <code>keepit_svil_bulk</code> , dedicato alle elaborazioni dell'ETL.
PostgreSQL	16	Conserva le informazioni interne di n8n, tra cui la configurazione, i workflow e la cronologia delle esecuzioni.

Tabella 5.4: Database utilizzati nel sistema

5.3.4 Infrastruttura

Tecnologia	Versione	Ruolo nel progetto
n8n	<code>stable</code>	Piattaforma sulla quale vengono definiti ed eseguiti i workflow ETL.
Docker	–	Esegue n8n e PostgreSQL in ambienti isolati chiamati contenitori.
Docker Compose	–	Configura e avvia insieme i contenitori di n8n e PostgreSQL e i relativi volumi.
Firebase Authentication	–	Verifica l'identità dell'Utente e gestisce le operazioni di accesso.
Redis	client 4.7.1	Conserva temporaneamente le informazioni sui permessi e può contribuire a limitare le richieste troppo frequenti.
Cloud Logging	–	Permette di raccogliere e consultare i log strutturati prodotti dal backend quando l'applicazione viene eseguita su Google Cloud Platform.

Tecnologia	Versione	Ruolo nel progetto
Gemini API	v1beta / 2.5 Flash	Propone le associazioni tra le colonne dei file CSV e i campi attesi nell'estensione dedicata al mapping.

Tabella 5.5: Tecnologie e servizi infrastrutturali

Redis non sostituisce il database applicativo, perché conserva soltanto informazioni temporanee. In questo progetto evita di rileggere dal database i permessi dell'Utente a ogni richiesta. Può inoltre condividere tra più istanze del backend il conteggio usato per limitare le richieste troppo frequenti.

5.3.5 Strumenti di sviluppo

Strumento	Versione	Ruolo nel progetto
pnpm	10.14.0	Installa le librerie del progetto ed esegue i comandi di backend e frontend.
Git e GitHub	–	Conservano la cronologia delle modifiche e permettono di condividere il codice.
Visual Studio Code	–	È l'ambiente usato per scrivere il codice.
Biome	2.3.10	Formatta il codice e segnala alcuni errori o problemi di stile.
tsx	4.22.3	Avvia il backend durante lo sviluppo e lo riavvia quando il codice cambia.
tsup	8.5.1	Prepara i file del backend da distribuire ed eseguire in produzione.
Vitest e Supertest	3.2.4 / 7.2.2	Eseguono i test automatici e controllano le risposte delle API.
Husky	9.1.7	Esegue gli hook Git usati per guidare e verificare i messaggi di commit.
lint-staged	15.5.2	Permette di applicare i controlli soltanto ai file preparati per il commit.
Commitlint e Commitizen	19.8.1 / 4.3.1	Aiutano a scrivere i messaggi di commit secondo un formato comune.
Sequelize Auto	0.8.8	Crea automaticamente i modelli TypeScript a partire dalle tabelle MySQL esistenti.

Tabella 5.6: Strumenti usati durante lo sviluppo

Husky esegue gli hook Git configurati nel progetto. Durante un commit, Commitizen aiuta a scrivere il messaggio nel formato scelto e Commitlint verifica che sia corretto. Biome controlla la formattazione e lo stile del codice, mentre lint-staged permette di applicare questi controlli soltanto ai file preparati per il commit. Sequelize Auto genera i modelli TypeScript a partire dalle tabelle MySQL, evitando di scriverli a mano.

5.4 Progettazione del backend e delle API

La progettazione del backend riguarda due aspetti: la suddivisione interna dei componenti e l'organizzazione delle API attraverso le quali il sistema espone le operazioni ETL.

5.4.1 Architettura logica del backend

L'architettura del backend è suddivisa in quattro livelli logici, ciascuno con una responsabilità precisa. Il livello di esposizione gestisce le richieste HTTP, l'autenticazione e il controllo dei dati ricevuti. Il livello applicativo coordina le operazioni richieste dall'Utente e la comunicazione con n8n. La logica di dominio raccoglie le regole proprie del processo ETL, in particolare quelle dedicate alla validazione. Infine, il livello di accesso ai dati isola le operazioni eseguite sui database.

La Figura 5.2 rappresenta questa suddivisione. Le frecce indicano le dipendenze consentite tra i livelli e le comunicazioni con i sistemi esterni, senza descrivere la sequenza delle fasi di importazione.

Nel codice, il livello di esposizione comprende il router e il controller, mentre il service coordina le operazioni. Il motore di validazione applica le regole sui dati e i repository gestiscono l'accesso ai database. La sezione successiva mostra come questi componenti sono collegati.

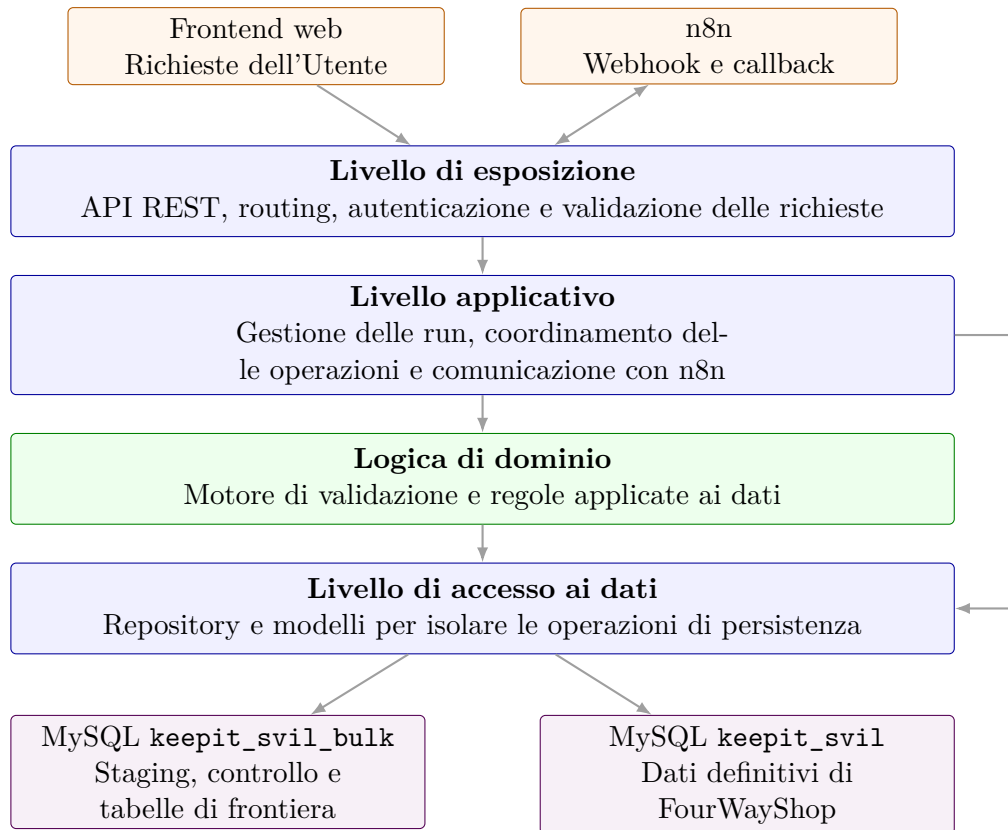


Figura 5.2: Architettura logica del backend del sistema ETL

5.4.2 Diagramma delle classi principali

La Figura 5.3 mostra i componenti appena descritti e i loro collegamenti. Per non rendere il diagramma troppo complesso, sono riportati solo i metodi principali. I tipi definiti in `etlModel.ts`, gli schemi Zod e le funzioni di supporto non sono invece rappresentati.

Il router collega gli *endpoint*^G ai metodi di `EtlController`. Il controller gestisce le richieste e le risposte HTTP e passa le operazioni a `EtlService`. Il service coordina i casi d'uso e richiama i repository necessari. `EtlValidationEngine` gestisce la validazione applicando le regole definite in `etlValidationRules`. Il repository delle correzioni usa il motore per controllare i dati inseriti dall'Utente, mentre `EtlLoadRepository` gestisce i controlli e il caricamento delle tabelle di destinazione.

I repository usano Sequelize per accedere ai database MySQL. `EtlCorrectionRepository` è l'unica eccezione, perché non interroga direttamente il database. Nel diagramma, `loadLookupSet()` indica i metodi che caricano i dati di riferimento, mentre `loadArtNN()` indica quelli dedicati alle tabelle `artXX`. Questi nomi sono abbreviazioni e non corrispondono a singoli metodi. Le frecce mostrano quali componenti vengono utilizzati e non indicano rapporti di ereditarietà.

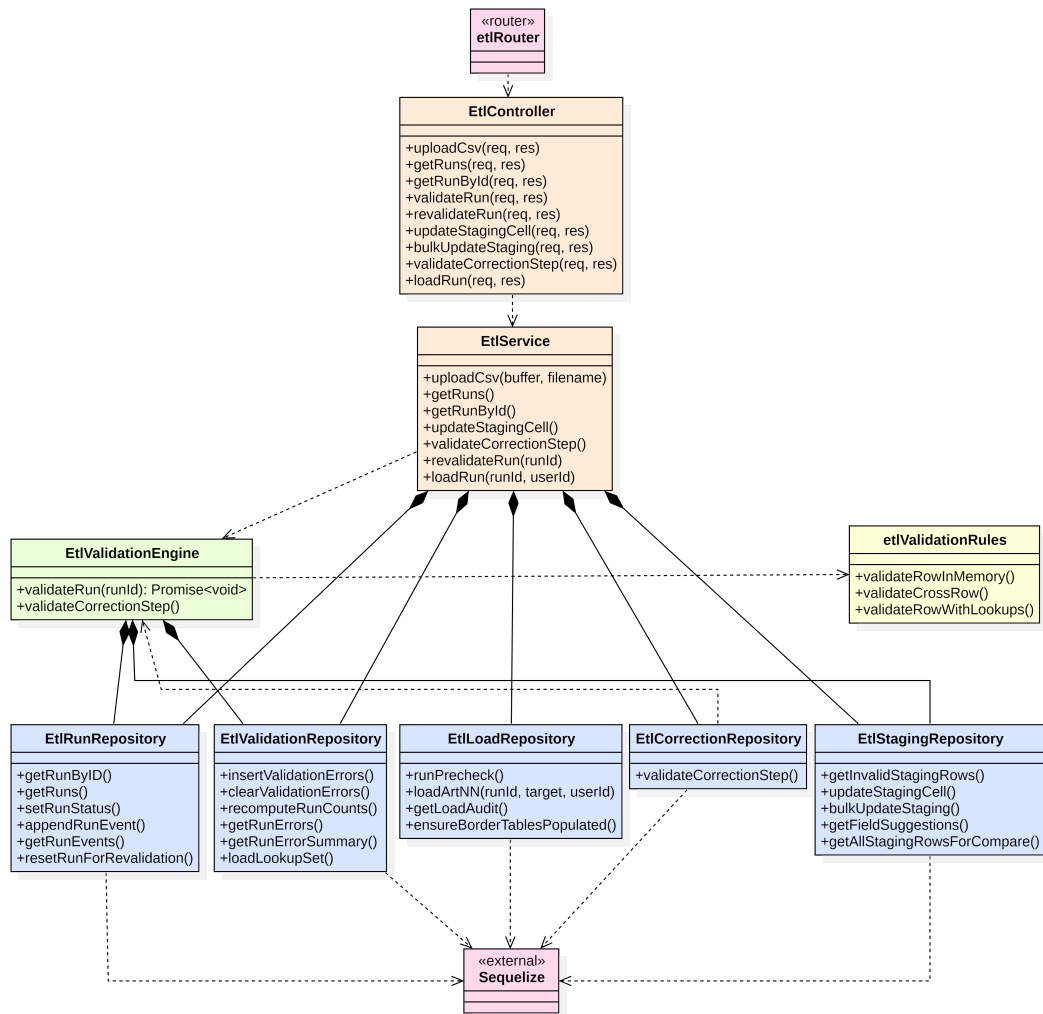


Figura 5.3: Diagramma delle principali classi e dei moduli del backend ETL

5.4.3 Organizzazione delle API

Le API ETL sono raccolte sotto il percorso `/api/v1/etl`. Quando un'operazione riguarda una specifica importazione, il percorso contiene anche l'identificativo della run. Le richieste `GET` recuperano i dati, le richieste `PATCH` modificano lo staging e le richieste `POST` avviano le diverse operazioni del processo.

La Tabella 5.7 riepiloga i principali gruppi di API. Per brevità, i percorsi omettono il prefisso comune `/api/v1/etl`; i puntini sostituiscono invece la parte `/runs/{runId}`.

Gruppo	Operazioni principali	Responsabilità
Avvio	<code>POST /runs</code>	Riceve il file CSV e avvia una nuova importazione.
Consultazione	<code>GET /runs</code> <code>GET /runs/{runId}</code> <code>GET /runs/stats</code>	Restituisce l'elenco delle run, il dettaglio di una singola esecuzione e le statistiche complessive.
Controllo	<code>GET .../errors</code> <code>GET .../error-summary</code> <code>GET .../events</code> <code>GET .../load-audit</code>	Espone il riepilogo e il dettaglio degli errori, la cronologia degli eventi e le informazioni sul caricamento finale.
Correzione	<code>GET .../staging/invalid</code> <code>PATCH .../staging/cell</code> <code>PATCH .../staging/bulk</code>	Recupera i record non validi e permette di correggere un singolo valore o un insieme di righe.
Supporto alla correzione	<code>GET .../field-suggestions</code> <code>POST .../corrections/validate-step</code>	Fornisce valori utili alla correzione e verifica il singolo passaggio della procedura guidata.
Prosecuzione del flusso	<code>POST .../revalidate</code> <code>POST .../load</code>	Avvia la rivalidazione completa oppure, dopo la conferma dell'Utente, il caricamento definitivo.
Callback di n8n	<code>POST .../validate</code> <code>POST .../populate-border</code>	Permette ai workflow di richiedere al backend la validazione e il popolamento delle tabelle di frontiera.

Tabella 5.7: Principali gruppi di API del modulo ETL

Le API usate dal frontend richiedono l'autenticazione dell'Utente. Le richieste

inviata direttamente da n8n non hanno invece un token utente e vengono riconosciute tramite l'intestazione `x-n8n-secret`. Il valore è configurato sia in n8n sia nel backend e la richiesta viene accettata solo se i due valori coincidono. Gli schemi Zod descrivono e controllano i dati in ingresso e in uscita. Gli stessi schemi vengono usati anche per generare la documentazione OpenAPI.

5.5 Progettazione del modello dati

Il modello dati del sistema ETL è stato progettato a partire dalla struttura di `keepit_svil`, il database finale di FourWayShop. In questo database le informazioni relative a un prodotto non sono contenute in un'unica tabella, ma sono distribuite tra diverse entità collegate tramite chiavi esterne. Le tabelle principali interessate dall'importazione seguono generalmente la nomenclatura `artXX`: oltre agli articoli comprendono, ad esempio, marche, stagioni, gerarchie, attributi, dimensioni, fornitori e codici a barre.

L'obiettivo del sistema è preparare i dati necessari a popolare correttamente queste tabelle. Per questo motivo, nel database `keepit_svil_bulk` sono state create strutture intermedie che riprendono le tabelle di destinazione. Il modello è completato da una tabella di staging e da alcune tabelle di supporto, necessarie per controllare le importazioni e conservarne la cronologia.

Le run e i principali record ETL sono identificati tramite `UUIDG` memorizzati in colonne `BINARY(16)`. Questo formato è stato imposto da Sinesy, che lo utilizzava già perché, nelle proprie prove interne, era risultato più veloce rispetto alla rappresentazione testuale. Occupa inoltre meno spazio e rende più compatti gli indici. Nelle tabelle di audit in cui è sufficiente un identificativo progressivo vengono invece usate chiavi numeriche.

5.5.1 Area di staging

La prima tabella coinvolta è `brd_items_stg`, che riceve i dati del file dopo la normalizzazione. Ogni riga contiene, in apposite colonne, i riferimenti alla run, all'azienda e alla posizione occupata nel file originale. Le restanti colonne memorizzano le informazioni relative al prodotto, tra cui articolo, marca, stagione, fornitore, gerarchie, attributi, dimensioni, prezzi e codici a barre.

In questa fase i dati non sono ancora considerati corretti. La tabella può quindi contenere valori mancanti, formati non validi o informazioni non coerenti con quelle già presenti in `keepit_svil`. Ogni record possiede uno stato di validazione, inizialmente `PENDING` e successivamente impostato a `VALID` oppure `INVALID`. Lo staging costituisce così un'area di lavoro separata, nella quale i dati possono essere controllati e corretti senza modificare il database finale.

5.5.2 Tabelle di controllo e tracciabilità

Accanto allo staging sono state introdotte alcune tabelle di supporto, mostrate nella Figura 5.4. I colori ne distinguono le funzioni: il blu identifica la gestione delle run e dei relativi eventi, il verde lo staging e l'audit delle correzioni, l'arancione gli errori di validazione e il viola il controllo del caricamento finale. La tabella `brd_integration_runs` contiene una riga per ogni importazione. Registra il nome del file ricevuto, l'azienda, lo stato corrente, i contatori dei blocchi e delle righe, gli istanti di inizio e fine e l'eventuale messaggio di errore. L'identificativo della run collega tutte le operazioni e i dati della stessa esecuzione.

La tabella `brd_integration_run_events` registra le attività svolte dal backend e dai workflow. Lo stato della run mostra la fase corrente, mentre gli eventi permettono di vedere i passaggi già eseguiti.

Gli errori trovati durante la validazione vengono salvati in `brd_validation_errors`. Ogni errore è collegato alla run, alla riga sorgente, al campo e al valore che lo hanno generato. La tabella registra anche il codice, il messaggio, la gravità e la fase bloccata dall'errore. Il frontend può così mostrare all'Utente sia un riepilogo sia le correzioni richieste.

Quando un dato viene corretto, `brd_items_stg_audit` registra il campo modificato, il valore precedente, quello nuovo, l'autore e l'istante della modifica. `brd_final_load_audit` registra invece l'esito dei tentativi di caricamento sulle singole tabelle, i conteggi dei record e gli eventuali errori. Il modello comprende anche `brd_final_load_log`, predisposta per registrare informazioni più sintetiche.

Queste tabelle mantengono lo stato dell'elaborazione nel database. Il frontend può quindi mostrare l'avanzamento della run, gli errori trovati e le correzioni eseguite anche quando il processo coinvolge componenti diversi.

brd_integration_runs - id # company_id - Az flow_code - Az source_filename - Az status # total_rows # chunks_total # chunks_processed # chunk_size # current_chunk_index # valid_rows # invalid_rows # warning_count 🕒 started_at 🕒 finished_at - Az error_message	brd_items_stg Identificazione - id # run_id # company_id # source_row_number Stagione - Az season_code - Az season_description_it - Az season # season_year Marca - Az brand_code - Az brand_description Fornitore - Az supplier_code - Az item_supplier ☒ main Articolo - Az item_code - Az item_description_it - Az item_shop_description_it Gerarchie - Az hierarchy_level1 - Az hierarchy_level1_description - Az hierarchy_level2 - Az hierarchy_level2_description - Az hierarchy_level3 - Az hierarchy_level3_description - Az hierarchy_level4 - Az hierarchy_level4_description - Az hierarchy_level5 - Az hierarchy_level5_description - Az hierarchy_level6 - Az hierarchy_level6_description - Az hierarchy_level7 - Az hierarchy_level7_description - Az hierarchy_level8 - Az hierarchy_level8_description - Az hierarchy_level9 - Az hierarchy_level9_description - Az hierarchy_level10 - Az hierarchy_level10_description Attributi - Az attribute_level1 - Az attribute_level1_description - Az attribute_level2 - Az attribute_level2_description - Az attribute_level3 - Az attribute_level3_description - Az attribute_level4 - Az attribute_level4_description - Az attribute_level5 - Az attribute_level5_description - Az attribute_level6 - Az attribute_level6_description - Az attribute_level7 - Az attribute_level7_description - Az attribute_level8 - Az attribute_level8_description - Az attribute_level9 - Az attribute_level9_description - Az attribute_level10 - Az attribute_level10_description
--	--

Gruppi di dimensione

- Az dimension_group1
- Az dimension_group2
- Az dimension_group3
- Az dimension_group4
- Az dimension_group5

Codice IVA e barcode

- Az vat_code
- Az barcode_type
- Az barcode

Livelli di dimensione

- Az dimension_level1
- Az dimension_level2
- Az dimension_level3
- Az dimension_level4
- Az dimension_level5

Prezzi

- # item_price_acq
- # item_price_fat
- Az item_price_ven

Stato e data

- ☒ validation_status
- 🕒 created_at

Figura 5.4: Tabelle di staging, controllo e tracciabilità

5.5.3 Tabelle di frontiera

Al termine della validazione, i record validi vengono suddivisi nelle tabelle di frontiera mostrate nella Figura 5.5. Queste strutture si trovano nel database `keepit_svil_bulk` e sono riconoscibili dal prefisso `brd_art`. Riprendono soltanto le tabelle di `keepit_svil` coinvolte nell'importazione.

La tabella `brd_art01_items` contiene i dati principali degli articoli. Le altre tabelle separano codici a barre, gerarchie, attributi, gruppi e livelli di dimensione, marche, stagioni e associazioni con i fornitori. In questo modo i dati inizialmente raccolti in un'unica riga di staging assumono una struttura simile a quella richiesta da `keepit_svil`.

A differenza delle tabelle definitive, le tabelle di frontiera non mantengono le relazioni tramite chiavi esterne. I collegamenti rimangono rappresentati attraverso codici applicativi, come il codice dell'articolo, della marca o della stagione. Questa scelta evita controlli relazionali durante il passaggio intermedio e rende più semplici le operazioni necessarie a suddividere e preparare i dati.

Le tabelle di frontiera devono contenere soltanto dati che hanno già superato la validazione. L'assenza di relazioni serve quindi a semplificare questo passaggio, non ad accogliere dati errati. I vincoli vengono applicati durante il caricamento in `keepit_svil`; se un'incoerenza non è stata rilevata in precedenza, il database finale restituisce un errore, che viene registrato e segnalato all'Utente.

brd_art02_item_barcode id # run_id # company_id Az item_code Az barcode_type Az barcode Az dimension_level1 Az dimension_level2 Az dimension_level3 Az dimension_level4 Az dimension_level5 created_at	brd_art01_items Identificazione id # run_id # company_id Az item_code Descrizioni Az item_description_it Az item_shop_description_it Collegamenti Az season_code Az season Az brand_code Az vat_code Gerarchie Az hierarchy_level1 Az hierarchy_level2 Az hierarchy_level3 Az hierarchy_level4 Az hierarchy_level5 Az hierarchy_level6 Az hierarchy_level7 Az hierarchy_level8 Az hierarchy_level9 Az hierarchy_level10 Attributi Az attribute_level1 Az attribute_level2 Az attribute_level3 Az attribute_level4 Az attribute_level5 Az attribute_level6 Az attribute_level7 Az attribute_level8 Az attribute_level9 Az attribute_level10 Gruppi dimensione Az dimension_group1 Az dimension_group2 Az dimension_group3 Az dimension_group4 Az dimension_group5 Prezzi # item_price_acq # item_price_fat # item_price_ven Data created_at	brd_art09_dimension_groups id # run_id # company_id # dimension_position Az dimension_group_code created_at
brd_art04_hierarchy_levels id # run_id # company_id # level_number Az hierarchy_code Az hierarchy_description Az parent_hierarchy_code created_at		brd_art12_brands id # run_id # company_id Az brand_code Az brand_description created_at
brd_art05_attribute_levels id # run_id # company_id # level_number Az attribute_code Az attribute_description Az parent_attribute_code created_at		brd_art28_seasons id # run_id # company_id Az season_code Az season_description_it Az season # season_year created_at
brd_art06_dimension_levels id # run_id # company_id # dimension_position Az dimension_group_code Az dimension_level_code created_at		brd_art35_item_suppliers id # run_id # company_id Az item_code Az supplier_code Az item_supplier # main created_at

Figura 5.5: Tabelle di frontiera delle entità da caricare in keepit_svil

5.6 Progettazione del mapping dei campi

Il mapping assistito dall'intelligenza artificiale è stato progettato come un'estensione successiva della soluzione principale ed è stato quindi sviluppato separatamente dal flusso originario basato su n8n. Il suo scopo è permettere al sistema di elaborare file CSV con intestazioni diverse da quelle previste, associando le colonne ricevute ai campi interni dell'ETL.

A ogni caricamento, il sistema calcola un *hash*^G delle intestazioni del file e lo usa per verificare se quella struttura è già conosciuta. Se trova un mapping confermato e compatibile con le colonne ricevute, lo riutilizza automaticamente; in caso contrario, Gemini propone una nuova corrispondenza tra le colonne del CSV e i campi attesi. La nuova proposta viene mostrata all'Utente, che può correggerla e confermarla prima di proseguire con l'elaborazione. In questo modo, l'intelligenza artificiale facilita la configurazione, ma la decisione finale rimane affidata all'Utente.

I mapping confermati vengono salvati nella tabella `brd_etl_column_mappings`, mostrata nella Figura 5.6. La configurazione non è legata a una singola run, poiché può essere riutilizzata ogni volta che viene caricato un file con la stessa struttura.

brd_etl_column_mappings	
Identificazione	
🔑	id
123	company_id
123	client_id
Riconoscimento del tracciato	
T≡	header_hash
T≡	source_headers_json
T≡	sample_signature
Configurazione	
T≡	mapping_json
T≡	provider
T≡	model
T≡	status
Date	
🕒	created_at
🕒	updated_at
🕒	last_used_at

Figura 5.6: Tabella utilizzata per memorizzare e riutilizzare i mapping

La tabella associa il mapping all'azienda e alle caratteristiche del tracciato. In pratica memorizza l'hash delle intestazioni, le colonne sorgenti e le associazioni confermate. Registra inoltre l'origine, lo stato, la data di creazione, l'ultimo aggiornamento e l'ultimo utilizzo della configurazione. Per ciascun tracciato viene mantenuto un solo mapping attivo nell'ambito della stessa azienda, così da evitare ambiguità durante il riutilizzo.

I dettagli relativi all'integrazione con Gemini e ai controlli eseguiti dal backend verranno descritti nel capitolo dedicato allo sviluppo.

5.7 Progettazione del flusso ETL

Il flusso ETL accompagna il file dal caricamento fino alla scrittura dei dati in `keepit_svil`. Il processo è suddiviso in fasi consecutive e, al termine di ciascuna, vengono salvati i risultati ottenuti e aggiornato lo stato della run.

5.7.1 Ricezione del file e creazione della run

Il processo inizia con il caricamento di un file CSV. Il backend genera l'identificativo dell'importazione e verifica che il file contenga un'intestazione e almeno una riga di dati.

Prima di affidare l'elaborazione a `n8n`, il backend divide il file in blocchi, evitando di trasferire in un'unica richiesta contenuti di grandi dimensioni. Ogni blocco conserva l'intestazione e le informazioni necessarie per ricostruirne la posizione nel CSV, compreso il numero della riga originale. Alla ricezione del primo blocco, il workflow registra la run nello stato `RECEIVED`; da quel momento l'importazione rimane consultabile anche dopo che l'Utente ha lasciato la pagina dalla quale l'ha avviata.

5.7.2 Controllo del tracciato e normalizzazione

Dopo la registrazione della run, il workflow controlla ciascun blocco ricevuto e verifica che il tracciato contenga tutte le colonne obbligatorie. Se una o più colonne sono assenti, la run passa allo stato `FAILED_NORMALIZATION`.

Superato il controllo, i dati vengono portati in una forma uniforme: gli spazi non significativi vengono rimossi, i campi vuoti vengono rappresentati come valori nulli e i dati numerici vengono convertiti nei tipi previsti. Vengono inoltre uniformati i valori che indicano vero o falso e applicati i limiti di lunghezza richiesti dallo staging.

I record normalizzati vengono quindi inseriti nella tabella `brd_items_stg` con stato `PENDING`, mantenendo il riferimento alla riga del file sorgente. Al com-

pletamento di ogni blocco viene aggiornato l'avanzamento della run; quando anche l'ultimo blocco è stato elaborato, la run passa allo stato **NORMALIZED** e può iniziare la validazione.

5.7.3 Validazione dei dati

La validazione analizza i dati presenti nello staging attraverso tre categorie di controlli:

1. **controlli sul singolo record**, che verificano, ad esempio, i campi obbligatori, la validità dell'anno, il formato dei valori e l'assenza di prezzi negativi;
2. **controlli trasversali al file**, che individuano duplicazioni o associazioni incoerenti, come lo stesso codice a barre collegato ad articoli diversi;
3. **controlli sui dati di riferimento**, che confrontano i codici ricevuti con fornitori, aliquote IVA, tipi di codice a barre e altre configurazioni presenti in `keepit_svil`.

Ogni errore viene registrato e collegato al record che lo ha generato. Una riga che presenta almeno un errore bloccante assume lo stato **INVALID**; se supera i controlli, passa invece allo stato **VALID**. Gli avvisi vengono conteggiati separatamente e non rendono necessariamente invalido il record. Al termine della fase, il sistema aggiorna i contatori e lo stato complessivo della run.

5.7.4 Correzione e rivalidazione

Se la validazione trova errori bloccanti, il sistema mostra una procedura guidata, o *wizard*^G. Gli errori sono raggruppati per campo e tipologia e vengono mostrati in un ordine preciso. In questo modo vengono affrontati prima quelli che possono influire sulle correzioni successive. L'Utente può correggere una singola riga oppure applicare lo stesso valore a più record quando la regola lo consente.

Ogni passaggio può mostrare i valori ammessi, quelli presenti nei dati di riferimento o altri valori coerenti trovati nel file. I suggerimenti non vengono applicati automaticamente: la scelta rimane sempre all'Utente. Prima di passare all'errore successivo, il sistema controlla di nuovo la regola e segnala se il problema è ancora presente.

Ogni correzione viene registrata nell'audit. Al termine della procedura guidata, il sistema valida di nuovo l'intera run, perché una modifica può influire anche sui controlli delle altre righe. La nuova validazione rimuove gli errori risolti e aggiorna lo stato dei record e i contatori. Finché rimane almeno una riga non valida, il caricamento finale non può iniziare.

5.7.5 Popolamento delle tabelle di frontiera

I dati validi presenti nello staging vengono trasferiti nelle rispettive tabelle di frontiera. Durante questa trasformazione, le informazioni contenute in ciascun record vengono separate per entità e le occorrenze duplicate vengono unificate in base alle relative chiavi.

Se tutte le righe hanno superato i controlli, la run raggiunge lo stato `BORDER_LOADED`: i dati sono pronti per il caricamento, ma `keepit_svil` non è ancora stato modificato. Se sono ancora presenti righe non valide, la run passa invece a `PARTIAL_SUCCESS` e il caricamento definitivo rimane bloccato.

5.7.6 Caricamento definitivo

Il caricamento in `keepit_svil` non viene avviato automaticamente. Il sistema mostra prima un riepilogo della run e richiede la conferma dell'Utente, che può così verificare i conteggi prima di modificare le tabelle definitive.

Dopo la conferma, un controllo preliminare, chiamato *precheck*, verifica i dati presenti nelle tabelle di frontiera e la disponibilità delle configurazioni necessarie. Gli eventuali problemi vengono registrati e impediscono l'avvio del caricamento. Su indicazione di Sinesy, il caricamento nelle tabelle di `keepit_svil` è stato sviluppato direttamente nel progetto, senza ricorrere a servizi preesistenti.

Superato il controllo, le entità vengono caricate rispettando le loro dipendenze. Stagioni, marche, gerarchie, attributi e dimensioni precedono gli articoli; fornitori e codici a barre vengono invece elaborati quando è disponibile l'identificativo dell'articolo. Per ciascuna entità, il sistema stabilisce se inserire un nuovo record oppure aggiornare quello già esistente.

L'esito di ogni passaggio viene registrato nelle tabelle di audit. La run raggiunge lo stato `SUCCESS` soltanto quando tutte le operazioni sono terminate correttamente; in caso contrario, vengono conservate le informazioni sulla tabella coinvolta e sul tentativo eseguito, così da rendere più semplice l'individuazione del problema.

5.7.7 Ciclo di vita della run

Il ciclo di vita permette di seguire l'avanzamento del processo. La Tabella 5.8 riporta gli stati principali.

Stato	Significato
RECEIVED	Il file è stato ricevuto e la run è stata creata.
NORMALIZING	I blocchi sono in fase di controllo e normalizzazione.

NORMALIZED	Tutti i blocchi sono stati inseriti nello staging.
VALIDATING	Sono in corso i controlli di validazione.
VALIDATED	La validazione è terminata e possono essere preparate le entità di frontiera.
FAILED_VALIDATION	Nessuna riga ha superato la validazione.
LOADING_BORDER	È in corso il popolamento delle tabelle di frontiera.
BORDER_LOADED	Tutti i dati hanno superato la validazione e sono pronti per il caricamento definitivo.
PARTIAL_SUCCESS	Il popolamento delle tabelle di frontiera ha considerato soltanto una parte delle righe, perché le altre non sono valide.
LOADING_FINAL	È in corso la scrittura nelle tabelle di <code>keepit_svil</code> .
SUCCESS	Il caricamento definitivo è terminato correttamente.
Altri stati FAILED	La run si è interrotta per un problema tecnico o di configurazione. Il suffisso indica la fase interessata, ad esempio normalizzazione, tabelle di frontiera o precheck.

Tabella 5.8: Stati principali della run

Negli stati `PARTIAL_SUCCESS` e `FAILED_VALIDATION`, l'Utente può correggere le anomalie e avviare una nuova validazione. Il caricamento definitivo è disponibile soltanto nello stato `BORDER_LOADED`; non è quindi previsto il caricamento parziale dei soli record validi.

Non è stato introdotto un controllo separato per gestire i passaggi tra gli stati. Questi seguono semplicemente l'ordine delle operazioni definito nel workflow: quando una fase termina correttamente, il processo passa a quella successiva; in caso contrario, la run assume lo stato di errore relativo alla fase e il flusso si interrompe. Per esempio, se la validazione fallisce, la run passa allo stato `FAILED_VALIDATION` e il caricamento delle tabelle di frontiera non viene avviato. Di conseguenza, nel normale svolgimento del workflow il passaggio da `FAILED_VALIDATION` a `LOADING_BORDER` non avviene, anche se non è presente un controllo generale sullo stato precedente della run.

5.8 Aspetti trasversali

La progettazione ha tenuto conto anche di alcuni aspetti comuni a tutto il sistema, come la protezione dei dati, la gestione degli errori, le prestazioni e la

separazione delle informazioni delle diverse aziende.

5.8.1 Integrità e gestione degli errori

Le tabelle usate durante l'elaborazione sono separate da quelle definitive, così eventuali errori non modificano direttamente i dati finali. Le scritture sono suddivise in transazioni locali: se una modifica fallisce, vengono annullate solo quelle eseguite nella stessa transazione.

Gli errori vengono distinti in due categorie. Quelli presenti nei dati sono collegati alla riga e al controllo che li ha rilevati, in modo che l'Utente possa correggerli. I problemi tecnici riguardano invece i workflow, l'accesso ai database o la configurazione del sistema e richiedono un intervento tecnico.

5.8.2 Prestazioni e gestione dei volumi

Il file viene elaborato a blocchi, così il sistema non deve gestire troppe righe in una sola operazione. La run tiene il conto dei blocchi totali e di quelli completati. La validazione inizia solo dopo che tutti i blocchi sono stati normalizzati.

Anche il caricamento definitivo procede per gruppi di record. Quando possibile, il sistema controlla più elementi con una sola interrogazione, riducendo il numero di accessi al database.

5.8.3 Sicurezza e isolamento dei tenant

Ogni run e i relativi dati, come eventi, record di staging ed errori, sono associati a un'azienda. Per ogni richiesta, il backend verifica che la run appartenga all'azienda corretta e che l'Utente abbia i permessi necessari. Conoscere l'identificativo di una run non è quindi sufficiente per accedere ai dati di un'altra azienda.

Capitolo 6

Sviluppo del sistema ETL

Questo capitolo descrive lo sviluppo del sistema ETL progettato nel capitolo precedente. Vengono presentati i workflow realizzati con n8n, il codice del backend, le pagine del frontend e le principali scelte effettuate durante l'implementazione. Nell'ultima parte vengono descritte anche la versione sviluppata senza n8n e l'aggiunta del mapping dei campi assistito dall'intelligenza artificiale.

I blocchi di codice riportati nel capitolo sono estratti dai file del progetto. Per mantenerli brevi, vengono mostrate soltanto le righe utili alla spiegazione e vengono omesse le parti non necessarie.

6.1 Punto di partenza e versioni realizzate

Per lo sviluppo, Sinesy ha fornito un repository con una base di progetto già predisposta, chiamata anche *boilerplate*^G. Il progetto di partenza comprendeva il backend e il frontend già configurati, il sistema di autenticazione e le principali tecnologie usate da FourWayShop. Erano inoltre già presenti i componenti grafici, i colori e le regole seguite da Sinesy per organizzare il codice e realizzare le interfacce.

Il sistema ETL non è stato quindi sviluppato partendo da un progetto vuoto, ma è stato aggiunto a questa base. Questo ha permesso di mantenere lo stesso stile e la stessa organizzazione di FourWayShop. Inoltre, se Sinesy decidesse di integrare il lavoro realizzato, potrebbe farlo in modo più semplice e veloce, senza dover adattare un progetto costruito con tecnologie o impostazioni diverse.

Le tre versioni realizzate sono state mantenute in *branch*^G Git separati. I branch successivi derivano dalla versione precedente e introducono una modifica precisa:

`main` $\longrightarrow$ `main-code-only` $\longrightarrow$ `ai-mapping`

Branch	Deriva da	Contenuto
main	–	Soluzione originale composta da n8n, backend TypeScript e frontend React. n8n svolge il ruolo di orchestratore.
main-code-only	main	Versione senza n8n. Le operazioni svolte dall'orchestratore vengono spostate nel backend TypeScript, mentre il frontend viene mantenuto.
ai-mapping	main-code-only	Aggiunge alla versione senza n8n il mapping dei campi assistito da Gemini.

Tabella 6.1: Organizzazione delle versioni del progetto

Il branch `main` contiene quindi il progetto iniziale. Dopo averlo completato, è stata realizzata la versione `main-code-only` per verificare quanto n8n rendesse più pratici lo sviluppo e la gestione del processo. Il mapping assistito dall'intelligenza artificiale è stato aggiunto soltanto in seguito nel branch `ai-mapping`. Le sezioni successive seguono lo stesso ordine.

6.2 Orchestrazione con n8n

Nella soluzione originale l'orchestrazione è stata divisa in due workflow. Il primo riceve i blocchi del CSV, li normalizza e li inserisce nello staging. Il secondo avvia la validazione e, quando sono presenti righe valide, coordina il popolamento delle tabelle di frontiera.

Questa scelta deriva dal fatto che le aziende clienti di Sinesy utilizzano template CSV diversi. L'idea iniziale prevedeva quindi di creare un workflow di normalizzazione per ogni azienda. Ciascun workflow avrebbe trasformato il proprio tracciato nello stesso modello di dati, permettendo poi di utilizzare un unico workflow di validazione per tutti i clienti.

Nel progetto è stato realizzato un solo workflow di normalizzazione, relativo al tracciato preso come riferimento. Se la soluzione fosse stata integrata nel sistema aziendale, sarebbe poi spettato a Sinesy aggiungere gli altri workflow necessari. La separazione permette inoltre di ripetere la validazione dopo una correzione senza dover elaborare nuovamente il file. Nella versione successiva senza n8n, il mapping assistito dall'intelligenza artificiale ha superato la necessità di preparare un workflow di normalizzazione per ogni tracciato, perché permette di adattare template diversi al modello di dati comune.

6.2.1 Comunicazione tra backend e n8n

Il frontend non comunica direttamente con n8n. Quando l'Utente avvia un'importazione, il backend assegna un identificativo alla run e divide il contenuto del CSV in blocchi da un massimo di 15.000 righe. Questo limite è stato scelto come compromesso tra la quantità di dati inviata e il numero di richieste necessarie. Blocchi troppo grandi avrebbero potuto sovraccaricare il sistema o rendere difficile l'invio, mentre blocchi troppo piccoli avrebbero richiesto molti più invii, aumentando i tempi di elaborazione. Per questo, 15.000 righe sono sembrate una dimensione adatta a entrambe le esigenze. Ogni blocco mantiene l'intestazione del file e viene inviato tramite una richiesta HTTP al webhook del workflow di normalizzazione.

Oltre alle righe del CSV, la richiesta contiene l'identificativo della run, l'azienda, la posizione del blocco, il numero totale di blocchi e l'offset delle righe. Queste informazioni permettono a n8n di collegare tutte le elaborazioni alla stessa run e di mantenere il numero originale di ogni riga. Il backend invia i blocchi uno alla volta, seguendo il loro ordine nel file.

La comunicazione avviene anche nella direzione opposta. Il workflow di validazione richiama il backend attraverso due servizi interni: uno esegue i controlli sui dati presenti nello staging, mentre l'altro popola le tabelle di frontiera. Queste operazioni sono state implementate nel backend perché richiedono diversi controlli e accessi al database. n8n si occupa quindi di avviarle e gestirne l'ordine.

Le due API interne non utilizzano l'autenticazione dell'Utente. Ogni chiamata deve però contenere un segreto condiviso nell'header `x-n8n-secret`. Il backend confronta il valore ricevuto con quello configurato nell'ambiente e rifiuta la richiesta quando i due valori non coincidono. Il Codice 6.1 mostra il controllo applicato alle richieste provenienti da n8n.

```

1  const checkN8nSecret = (
2    req: Request,
3    res: Response,
4    next: NextFunction,
5  ) => {
6    const secret = req.headers["x-n8n-secret"];
7
8    if (secret !== env.N8N_INTERNAL_SECRET) {
9      res.status(401).json({
10        success: false,
11        message: "Unauthorized",
12      });
13      return;
14    }
15    next();
16  };
17
18  router.post("/etl/runs/:runId/validate", checkN8nSecret,
19    etlController.validateRun);
20  router.post("/etl/runs/:runId/populate-border", checkN8nSecret,
21    etlController.populateBorderTables);

```

Codice 6.1: Controllo delle chiamate interne provenienti da n8n

6.2.2 Workflow di normalizzazione

Il workflow di normalizzazione è mostrato nella Figura 6.1. Il nodo *Webhook* riceve il blocco inviato dal backend, mentre *Edit Fields* prepara i valori utilizzati dai nodi successivi. La query *RECEIVED* crea la run quando viene elaborato il primo blocco; per quelli successivi aggiorna la stessa esecuzione. Subito dopo, lo stato passa a **NORMALIZING**.

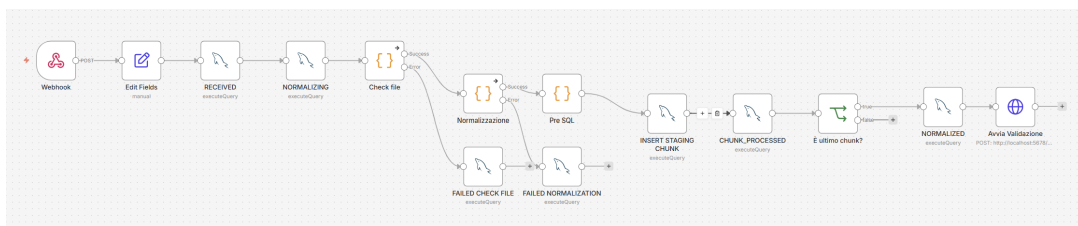


Figura 6.1: Workflow n8n per la normalizzazione

Il nodo *Check file* controlla che il blocco contenga l'intestazione e almeno una riga. Verifica inoltre la presenza delle colonne obbligatorie. Se il controllo fallisce, il ramo dedicato registra lo stato di errore e interrompe l'esecuzione del blocco.

Quando il tracciato è corretto, il nodo *Normalizzazione* legge le righe del CSV e associa ogni colonna al campo previsto nello staging. Durante questa operazione vengono rimossi gli spazi superflui, le stringhe vuote vengono trasformate in

valori nulli e i numeri vengono convertiti nel formato corretto. L'offset ricevuto dal backend viene usato per calcolare `source_row_number`, così gli eventuali errori successivi possono essere ricondotti alla riga originale.

Il nodo *Pre SQL* prepara un'unica istruzione di inserimento per tutte le righe del blocco, che viene poi eseguita da *INSERT STAGING CHUNK*. Al termine vengono aggiornati il numero di blocchi completati e l'indice dell'ultimo blocco elaborato.

Il nodo *È ultimo chunk?* stabilisce se sono ancora presenti blocchi da ricevere. Solo dopo l'ultimo, la run passa a *NORMALIZED* e il nodo *Avvia Validazione* richiama il secondo workflow. Un errore durante la conversione o l'inserimento segue invece il ramo *FAILED NORMALIZATION*.

6.2.3 Workflow di validazione e caricamento nelle tabelle di frontiera

Il secondo workflow, riportato nella Figura 6.2, viene avviato al termine della normalizzazione oppure quando l'Utente richiede una nuova validazione dopo aver corretto i dati.

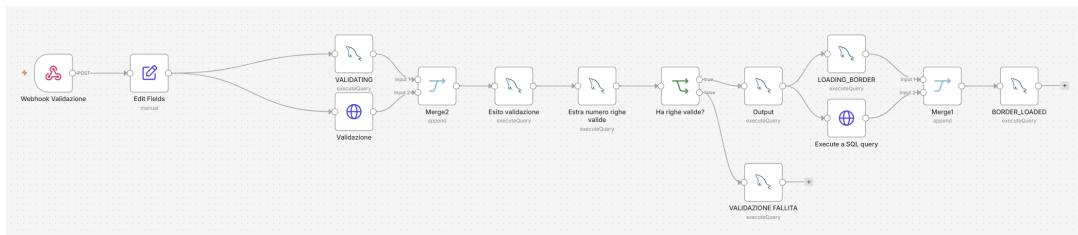


Figura 6.2: Workflow n8n per la validazione e le tabelle di frontiera

Dopo il nodo *Webhook Validazione*, il flusso aggiorna lo stato a *VALIDATING* e richiama il servizio di validazione del backend. Il nodo *Merge2* attende il completamento delle due operazioni prima di proseguire. La query successiva aggiorna i conteggi delle righe valide, non valide e degli avvisi.

Il nodo *Ha righe valide?* sceglie il ramo da eseguire. Se nessuna riga ha superato i controlli, la run passa a *FAILED_VALIDATION* e il workflow termina. Se è presente almeno una riga valida, il processo aggiorna lo stato a *LOADING_BORDER* e richiama il servizio del backend che prepara le tabelle di frontiera.

Anche in questo caso il nodo *Merge1* attende sia l'aggiornamento dello stato sia la risposta del backend. Al termine, la run raggiunge *BORDER_LOADED* quando tutte le righe sono valide oppure *PARTIAL_SUCCESS* quando rimangono errori da correggere. In quest'ultimo caso il caricamento definitivo non viene reso disponibile.

Il compito di n8n termina con la preparazione delle tabelle di frontiera. Il caricamento nel database definitivo non fa parte dei workflow: viene avviato dall'Utente e viene eseguito direttamente dal backend, come descritto nella Sezione 6.3.5.

6.3 Implementazione del backend

Il backend è stato sviluppato in TypeScript e contiene le operazioni che non vengono eseguite direttamente dai workflow n8n. Tra queste rientrano la gestione delle richieste provenienti dal frontend, la validazione dei dati, la correzione dello staging e il caricamento nelle tabelle definitive. Il backend espone inoltre i servizi interni richiamati da n8n durante la validazione e il popolamento delle tabelle di frontiera.

Le rotte ETL usano il controllo di autenticazione già presente nel boilerplate. Nel prototipo, l'azienda utilizzata dal modulo viene letta dalla configurazione `ETL_COMPANY_ID`; questo valore viene associato alle run e usato nelle query che elaborano i relativi dati. Durante l'integrazione con FourWayShop, lo stesso valore potrà essere ricavato dall'Utente autenticato.

6.3.1 Struttura del modulo ETL

Le API del sistema ETL sono definite nel router. Per ogni endpoint vengono indicati il metodo HTTP, il percorso, i dati richiesti e il tipo di risposta. Gli schemi Zod vengono usati per controllare i parametri ricevuti e per descrivere gli stessi servizi nella documentazione OpenAPI.

Il controller svolge controlli semplici sulle richieste: durante il caricamento, ad esempio, verifica che sia presente un file e che abbia estensione CSV. Per le altre operazioni legge l'identificativo della run, controlla il contenuto della richiesta tramite gli schemi previsti e richiama il metodo corrispondente del service.

Il service coordina le operazioni principali. Avvia i workflow di n8n, recupera i dati da mostrare nel frontend e richiama i repository necessari. I repository eseguono invece le interrogazioni tramite Sequelize e mantengono separati i diversi compiti:

- il repository delle run gestisce stati, conteggi ed eventi;
- il repository di staging legge e modifica i dati temporanei;
- il repository di validazione salva gli errori e aggiorna l'esito delle righe;
- il repository di caricamento prepara le tabelle di frontiera e aggiorna quelle definitive.

Questa divisione ha permesso di mantenere nel service soltanto il coordinamento del processo. Le query più lunghe, soprattutto quelle necessarie per il caricamento definitivo, rimangono nei repository dedicati.

6.3.2 Validazione dei dati

I tre tipi di controllo sono già stati presentati nella Sezione 5.7.3. Nel codice, la validazione è divisa tra tre componenti:

- `etlValidationRules.ts` contiene le singole regole;
- `EtlValidationEngine` ne coordina l'esecuzione;
- `EtlValidationRepository` salva gli errori e aggiorna lo staging.

Ogni regola controlla i dati e restituisce gli errori trovati, senza modificare direttamente il database. Una funzione prepara ogni errore indicando la riga del CSV, il campo, il codice, il messaggio, la gravità e il valore errato. Il Codice 6.2 mostra due controlli sull'anno della stagione e sul prezzo di acquisto.

```
1  if (row.season_year == null)
2    errors.push(err(row, "YEAR", "SEASON_YEAR_REQUIRED",
3      "YEAR obbligatorio.", null));
4  else if (row.season_year < 1900 || row.season_year > 2100)
5    errors.push(err(row, "YEAR", "SEASON_YEAR_INVALID",
6      "YEAR deve essere un anno plausibile.",
7      String(row.season_year)));
8
9  if (row.item_price_acq != null && row.item_price_acq < 0)
10    errors.push(err(row, "ITEM_PRICE_ACQ", "NEGATIVE_PRICE_ACQ",
11      "ITEM_PRICE_ACQ non può essere negativo.",
12      String(row.item_price_acq)));
```

Codice 6.2: Esempio di regole applicate a una singola riga

Per i controlli tra più righe vengono usate strutture `Map` e `Set`. Ad esempio, il codice collega ogni barcode all'insieme degli articoli nei quali compare. Se lo stesso barcode è collegato a più articoli, viene creato un errore per tutte le righe interessate. In questo modo non serve confrontare una riga alla volta con tutte le altre.

Anche i dati di riferimento vengono letti una sola volta. Fornitori, aliquote IVA, tipi di barcode e gruppi dimensionali vengono caricati insieme tramite `Promise.all` e salvati in alcuni insiemi. Le regole possono così controllare i codici senza interrogare ogni volta il database.

Il Codice 6.3 mostra la parte principale del motore. Prima vengono eseguite tutte le regole. In seguito, dagli errori bloccanti vengono ricavati i numeri delle righe non valide. Il repository usa questo elenco per aggiornare lo staging e i conteggi della run.

```
1  const allErrors: EtlValidationError[] = [];  
2  for (const row of rows) {  
3      allErrors.push(...validateRowInMemory(row));  
4  }  
5  allErrors.push(...validateCrossRow(rows));  
6  
7  const refSite = await this.validationRepository.resolveRefSite();  
8  const [supplierSet, vatCodeSet, barcodeTypeSet,  
9      dimensionGroupSet] = await Promise.all([  
10     this.validationRepository.loadSupplierSet(refSite.companyId),  
11     this.validationRepository.loadVatCodeSet(refSite.companyId),  
12     this.validationRepository.loadBarcodeTypeSet(refSite.companyId),  
13     this.validationRepository.loadDimensionGroupSet(  
14         refSite.companyId, refSite.siteId),  
15     ]);  
16  const lookups = { supplierSet, vatCodeSet,  
17     barcodeTypeSet, dimensionGroupSet };  
18  for (const row of rows) {  
19     allErrors.push(...validateRowWithLookups(row, lookups));  
20  }  
21  
22  const invalidRowNumbers = [...new Set(  
23     allErrors  
24         .filter((error) => error.severity === "ERROR")  
25         .map((error) => error.sourceRowNumber),  
26  )];  
27  
28  await this.validationRepository.insertValidationErrors(runId, allErrors  
29     );  
29  await this.validationRepository.updateStagingValidationStatus(  
30     runId, invalidRowNumbers);
```

Codice 6.3: Esecuzione delle regole e aggiornamento dello staging

Gli errori vengono salvati a gruppi, usando un'unica istruzione `INSERT` per ogni gruppo. Anche lo stato delle righe viene aggiornato a blocchi all'interno di una transazione. Alla fine vengono ricalcolati il numero di righe valide, quelle non valide e gli avvisi. Questi valori determinano anche lo stato della run.

6.3.3 Correzione e rivalidazione

I campi che l'Utente può correggere sono elencati nell'array `EDITABLE_STAGING_FIELDS`. Prima di eseguire la modifica, il backend controlla che il campo ricevuto sia presente in questo elenco e converte il nuovo valore nel tipo corretto. Una richiesta non può quindi modificare altri campi dello staging.

Quando viene modificata una cella, il repository legge prima il valore precedente con `SELECT FOR UPDATE`. Poi aggiorna lo staging e registra la modifica nella tabella di audit. Le due operazioni sono nella stessa transazione: se una fallisce, viene annullata anche l'altra. La modifica di più righe funziona nello stesso modo, ma prepara insieme tutte le righe da salvare nell'audit.

Per alcuni campi il backend propone anche dei valori utili alla correzione. I suggerimenti vengono letti dalle tabelle definitive e riguardano, ad esempio, fornitori, aliquote IVA, tipi di barcode, gerarchie, attributi e dimensioni. L'Utente può scegliere uno dei valori mostrati oppure inserirne uno diverso.

Per controllare una correzione, il backend usa il codice dell'errore per scegliere le regole da ripetere. Se l'errore riguarda un solo record, vengono controllate soltanto le righe modificate. Se dipende dal confronto tra più record, come nel caso dei barcode duplicati, vengono controllate tutte le righe della run. Prima del controllo vengono rimossi soltanto gli errori dello step corrente, mentre gli altri rimangono salvati.

6.3.4 Popolamento delle tabelle di frontiera

Il metodo `ensureBorderTablesPopulated` si occupa di popolare le tabelle di frontiera. Tutte le query vengono eseguite nella stessa transazione. Prima vengono eliminati gli eventuali dati già creati per la run, poi le tabelle vengono popolate di nuovo usando lo staging. In questo modo non rimangono dati del tentativo precedente.

Ogni tabella di frontiera viene popolata con una query `INSERT SELECT`. La query legge soltanto le righe dello staging con stato `VALID` e raggruppa i dati in base alla tabella da creare. Il Codice 6.4 mostra il caso dei fornitori associati agli articoli.

```
1 INSERT INTO brd_art35_item_suppliers
2   (run_id, company_id, item_code, supplier_code,
3    item_supplier, main)
4 SELECT x.run_id, x.company_id, x.item_code,
5        x.supplier_code, x.item_supplier,
6        COALESCE(NULLIF(TRIM(x.main), ''), 'F')
7 FROM (
8   SELECT s.*,
9          ROW_NUMBER() OVER (
10             PARTITION BY s.run_id, s.company_id,
11                          s.item_code, s.supplier_code
12             ORDER BY s.source_row_number
13          ) AS rn
14 FROM brd_items_stg s
15 WHERE s.run_id = UUID_TO_BIN(:rid)
16       AND s.company_id = :c
17       AND s.validation_status = 'VALID'
18       AND NULLIF(TRIM(s.item_code), '') IS NOT NULL
19       AND NULLIF(TRIM(s.supplier_code), '') IS NOT NULL
20 ) x
21 WHERE x.rn = 1;
```

Codice 6.4: Creazione delle associazioni tra articoli e fornitori

La funzione `ROW_NUMBER` numera le righe con lo stesso articolo e lo stesso fornitore. La query mantiene soltanto la prima, evitando di inserire più volte la stessa associazione. Marche e stagioni vengono raggruppate in modo simile. Per gerarchie, attributi e dimensioni vengono invece usate query con `UNION ALL`, che separano i diversi livelli. Se una query fallisce, la transazione annulla anche le operazioni precedenti.

6.3.5 Caricamento definitivo

Il caricamento definitivo è coordinato dal metodo `loadRun` di `EtlService`. Prima di modificare `keepit_svil`, il metodo esegue il precheck. Questo controllo verifica che le tabelle di frontiera contengano i dati da caricare e che siano presenti le configurazioni necessarie per gerarchie, attributi e dimensioni. Se trova un problema, il caricamento non inizia e la run passa allo stato di errore. Superato il precheck, il service individua l'azienda e il sito da usare per ogni tabella. Prepara inoltre alcune mappe che collegano i codici presenti nelle tabelle di frontiera agli identificativi usati in `keepit_svil`.

Le funzioni vengono eseguite nell'ordine mostrato nel Codice 6.5. Alcune usano direttamente il risultato di quelle precedenti. Ad esempio, `loadArt01` restituisce gli identificativi degli articoli, necessari per caricare in seguito i fornitori e i barcode.

```
1  const r28 = await withLog('art28_seasons',
2    () => repo.loadArt28(runId, tArt28, userId));
3  const r12 = await withLog('art12_brands',
4    () => repo.loadArt12(runId, tArt12, userId));
5  const r04 = await withLog('art04_hierarchy_levels',
6    () => repo.loadArt04(runId, tArt04, levelMap, userId));
7  const r05 = await withLog('art05_attribute_levels',
8    () => repo.loadArt05(runId, tArt05, positionMap, userId));
9  const r06 = await withLog('art06_dimension_levels',
10   () => repo.loadArt06(runId, tArt06, art09Map, userId));
11
12  const r01 = await withLog('art01_items',
13    () => repo.loadArt01(runId, tArt01, {
14      art04Cache, art05Cache: new Map(), art09Map, vatMap,
15      art28Cache: new Map(), art12Cache: new Map(),
16      levelMap, positionMap,
17    }, userId));
18  await withLog('art35_item_suppliers',
19    () => repo.loadArt35(runId, tArt35,
20      r01.art01ProgIdCache, userId));
21  await withLog('art02_item_barcodes',
22    () => repo.loadArt02Art47(runId, tArt02,
23      r01.art01ProgIdCache, art09Map,
24      r06.art06ProgIdCache, barcodeTypeMap, userId));
```

Codice 6.5: Ordine delle funzioni di caricamento definitivo

Ogni funzione legge la propria tabella di frontiera e divide i record in blocchi da 10.000 elementi. Per ogni blocco viene eseguita una sola query per cercare i codici già presenti nella tabella definitiva. I risultati vengono salvati in una mappa, usata per separare i record nuovi da quelli da aggiornare. Il Codice 6.6 mostra una versione semplificata di questo passaggio.

```
1  const CHUNK = 10_000;
2
3  for (let off = 0; off < border.length; off += CHUNK) {
4      const chunk = border.slice(off, off + CHUNK);
5
6      await sequelize.transaction(async tx => {
7          const existing = await sequelize.query<{
8              prog_id: number; season_code: string;
9          }>(
10             `SELECT prog_id, season_code
11                FROM keepit_svil.art28_seasons
12                WHERE company_id = :c AND site_id = :s
13                   AND season_code IN (:codes)`,
14             { replacements: { c: target.companyId,
15                             s: target.siteId,
16                             codes: chunk.map(row => row.season_code) },
17               type: QueryTypes.SELECT, transaction: tx },
18          );
19          const existMap = new Map(existing.map(
20              row => [row.season_code, row.prog_id]));
21
22          const toIns = chunk.filter(
23              row => !existMap.has(row.season_code));
24          const toUpd = chunk.filter(
25              row => existMap.has(row.season_code));
26
27          const ids = await this.reserveIds(
28              "art28_seasons", toIns.length, tx);
29          // Inserimento dei nuovi record e aggiornamento degli esistenti
30          .
31      });
32  }
```

Codice 6.6: Suddivisione tra record da inserire e da aggiornare

Il codice cambia leggermente per ogni tabella, ma il procedimento rimane lo stesso. Prima degli inserimenti vengono riservati i nuovi `prog_id`, aggiornando la sequenza globale nella stessa transazione. Gli articoli vengono inseriti a gruppi di 200. Per gli aggiornamenti più grandi, invece, viene usata una tabella temporanea e viene eseguita un'unica query di UPDATE.

Ogni funzione di caricamento viene richiamata tramite `withLog`. Questa funzione registra l'inizio dell'operazione e salva il numero di righe inserite e aggiornate. Se si verifica un errore, registra il fallimento e interrompe il caricamento. La run passa a `SUCCESS` soltanto dopo il completamento di tutte le tabelle.

6.4 Implementazione del frontend

Nel frontend sono state implementate le pagine necessarie per gestire il flusso ETL: caricamento del CSV, controllo dell'avanzamento, correzione degli errori e avvio del caricamento finale. Le pagine sono state sviluppate in React usando i componenti grafici già presenti nel boilerplate di Sinesy, così da mantenere lo stesso stile di FourWayShop.

Le chiamate al backend sono raccolte in un unico modulo. Ogni risposta viene controllata con gli schemi Zod prima di essere usata dal frontend. Se manca un dato o il formato non corrisponde a quello previsto, l'errore viene rilevato prima che la risposta sia mostrata nella pagina. Le sezioni successive descrivono le principali pagine realizzate.

6.4.1 Home del sistema ETL

La home riunisce le informazioni principali del sistema ETL. Nella parte superiore mostra il numero totale delle run e quante sono state completate, terminate parzialmente o fallite. La stessa pagina permette di avviare velocemente un nuovo import, aprire lo storico e consultare le run più recenti con il relativo stato e il numero di righe elaborate.

La Figura 6.3 mostra la home realizzata. Questa pagina permette all'Utente di controllare subito la situazione generale senza dover aprire separatamente l'elenco delle esecuzioni.

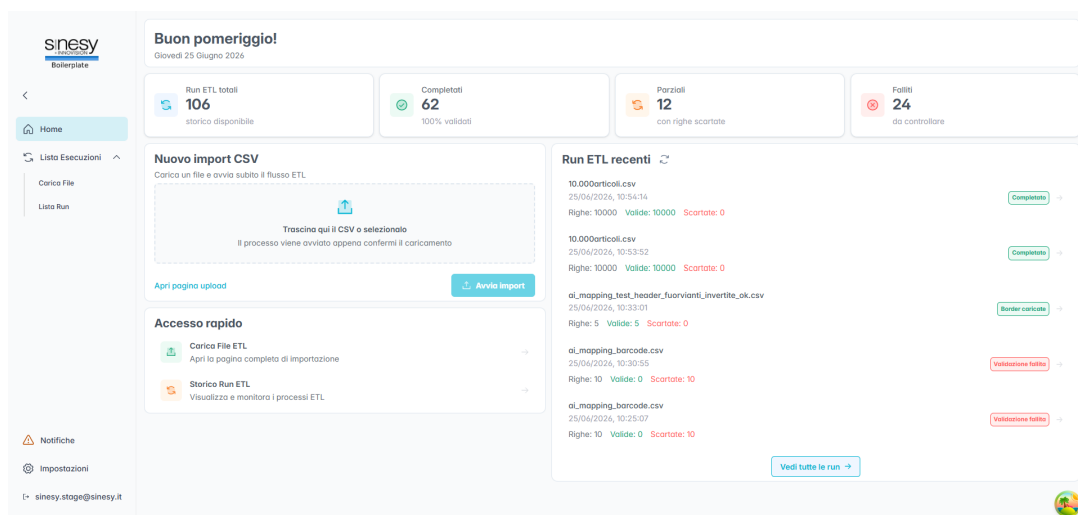


Figura 6.3: Home del sistema ETL

6.4.2 Caricamento del file

La pagina di caricamento permette di scegliere il CSV dal computer oppure di trascinarlo nell'area dedicata. Il frontend controlla l'estensione del file e mostra il nome e la dimensione prima dell'invio. Il pulsante di avvio rimane disabilitato finché non viene selezionato un file valido.

Quando l'Utente conferma, il frontend prepara una richiesta HTTP contenente il CSV e la invia al backend. La risposta contiene l'identificativo della nuova run; il frontend usa questo valore per aprire direttamente la pagina di dettaglio e mostrare l'avanzamento dell'elaborazione.

La Figura 6.4 mostra la pagina realizzata. Oltre all'area di caricamento, sono presenti il collegamento al template CSV di esempio e una breve spiegazione delle fasi principali.

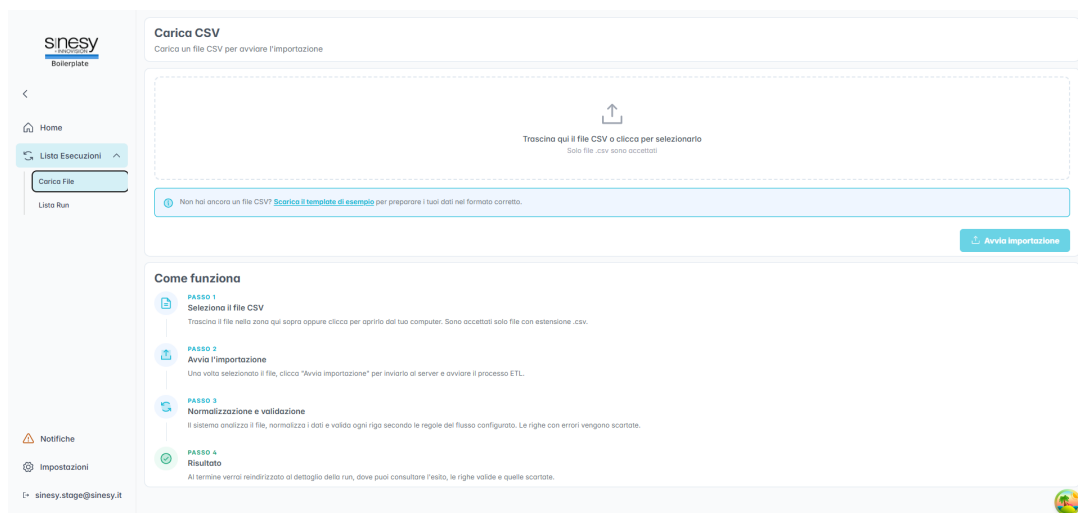


Figura 6.4: Pagina per il caricamento del file CSV

6.4.3 Consultazione delle esecuzioni

La pagina delle esecuzioni, mostrata nella Figura 6.5, riporta lo storico delle run in una tabella. Per ogni elemento vengono indicati il nome del file, lo stato, il numero di righe e gli orari di inizio e fine. È possibile filtrare i risultati per stato, nome del file e intervallo di date. Selezionando una riga si apre il relativo dettaglio.

La pagina di dettaglio rappresenta il processo con sei passaggi: ricezione, normalizzazione, validazione, caricamento nelle tabelle di frontiera, completamento delle tabelle di frontiera e caricamento nel database definitivo. Il passaggio corrente cambia colore in base allo stato della run.

Finché l'elaborazione è in corso, il frontend richiede al backend un aggiornamento ogni tre secondi. Quando viene raggiunto uno stato finale, l'aggiornamento

File	Stato	Righe	Inizio	Fine
10.000articoli.csv	Completato	10000	25/04/2026, 10:54:14	25/04/2026, 10:59:35
10.000articoli.csv	Completato	10000	25/04/2026, 10:53:52	25/04/2026, 10:58:42
ai_mapping_test_header_fuorvianti_invertite_ok.csv	Border caricato	5	25/04/2026, 10:33:01	25/04/2026, 10:33:03
ai_mapping_barcode.csv	Validazione fallita	10	25/04/2026, 10:30:55	25/04/2026, 10:30:57
ai_mapping_barcode.csv	Validazione fallita	10	25/04/2026, 10:25:07	25/04/2026, 10:25:08
partiti.csv	Completato	16	25/04/2026, 10:09:35	25/04/2026, 10:17:58
partiti.csv	Validazione fallita	16	25/04/2026, 09:28:50	25/04/2026, 09:29:06
10.000articoli.csv	Border caricato	10000	25/04/2026, 09:28:21	25/04/2026, 09:29:06
ai_mapping_sistemato.csv	Completato	10	23/04/2026, 09:21:14	23/04/2026, 09:22:02
ai_mapping_barcode.csv	Validazione fallita	10	22/04/2026, 09:21:14	22/04/2026, 09:22:02

Figura 6.5: Elenco delle esecuzioni ETL

automatico viene interrotto. La pagina mostra inoltre il numero di righe totali, valide e non valide, gli avvisi e il riepilogo degli errori. I messaggi tecnici restano in una sezione separata, in modo da non appesantire le informazioni principali mostrate all’Utente.

La Figura 6.6 mostra una run nella quale la validazione è fallita. La schermata riporta il punto in cui il processo si è fermato, il numero di righe non valide, il messaggio generale e il riepilogo degli errori. Per ogni errore sono mostrati il campo, il codice, il valore ricevuto e il relativo messaggio.

N. errori	Campo	Codice errore	Valore	Messaggio
10	BARCODE	BARCODE_REQUIRED		BARCODE obbligatorio.
1	BARCODE_TYPE	BARCODE_TYPE_NOT_FOUND	ZX99	BARCODE_TYPE non trovato.
1	DIMENSION_GROUP1	DIMENSION_GROUP1_NOT_FOUND	ZZZ_DIM_G_001	DIMENSION_GROUP1 non trovato in art09.
1	DIMENSION_LEVEL1	DIMENSION_LEVEL1_WITHOUT_GROUP1	M	DIMENSION_LEVEL1 richiede DIMENSION_GROUP1.
1	HIERARCHY_LEVEL2	HIERARCHY_LEVEL2_WITHOUT_LEVEL1	UOMO	HIERARCHY_LEVEL2 richiede HIERARCHY_LEVEL1.
1	SUPPLIER_CODE	SUPPLIER_NOT_FOUND	SUP-UNKNOWN-001	SUPPLIER_CODE non trovato in crm01Subjects.
1	VAT_CODE	VAT_CODE_NOT_FOUND	VAT-UNKNOWN-001	VAT_CODE non trovato in in36_vats.

Figura 6.6: Dettaglio di una run con errori di validazione

6.4.4 Correzione degli errori

Quando la validazione trova degli errori, dalla pagina di dettaglio è possibile aprire la procedura di correzione. Gli errori vengono raggruppati in step in base

al campo da modificare e ordinati secondo una priorità definita nel frontend. In questo modo l'Utente affronta un problema alla volta senza dover cercare manualmente le righe interessate.

Per ogni step vengono mostrati il messaggio di errore, le righe coinvolte, il valore attuale e gli eventuali suggerimenti. L'Utente può correggere una singola riga oppure applicare lo stesso valore a tutte le righe filtrate. La Figura 6.7 mostra la schermata completa del wizard, con l'avanzamento degli step, i suggerimenti, la modifica massiva e la correzione della singola riga.

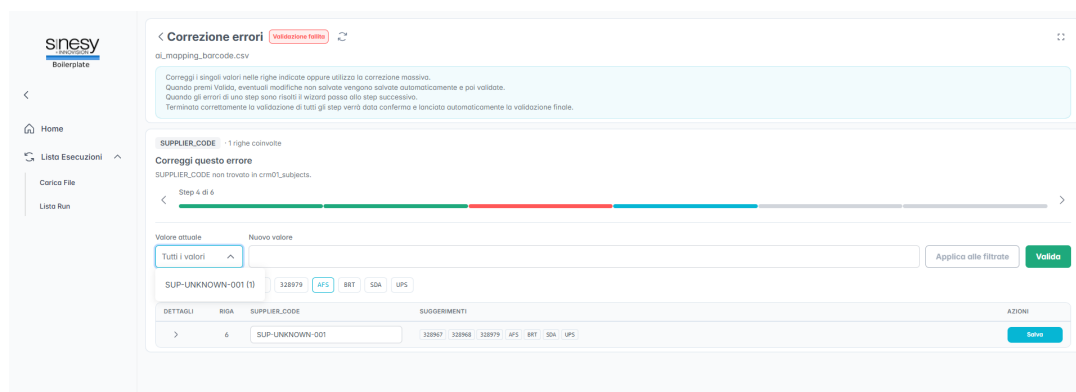


Figura 6.7: Procedura guidata per la correzione degli errori

Il pulsante di validazione dello step salva prima le eventuali modifiche ancora presenti nella pagina e richiede poi al backend un controllo mirato. Se l'errore è stato risolto, lo step viene segnato come completato e la procedura passa al successivo. Quando non rimangono errori, l'Utente può avviare la rivalidazione completa della run.

6.4.5 Caricamento nel database definitivo

Dopo il completamento della validazione e delle tabelle di frontiera, la pagina mostra il pulsante per avviare il caricamento nel database definitivo. Questa operazione viene quindi confermata dall'Utente e non parte automaticamente al termine delle fasi precedenti.

Durante il caricamento è possibile aprire i dettagli tecnici e seguire separatamente l'esito di ogni tabella. La Figura 6.8 riporta un esempio nel quale alcune tabelle sono già state completate e quella degli articoli è ancora in elaborazione.

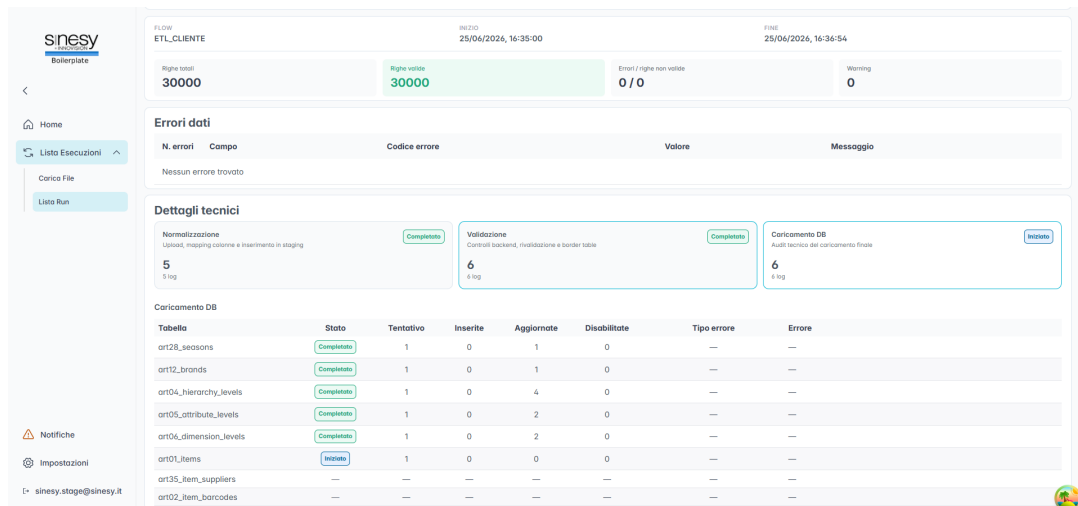


Figura 6.8: Avanzamento del caricamento nelle tabelle definitive

6.5 Implementazione della versione senza n8n

La versione senza n8n è stata sviluppata dopo aver completato la soluzione originale. Lo scopo era capire se n8n desse un valore aggiunto riconoscibile oppure se potesse essere sostituito senza particolari difficoltà.

Per rendere utile il confronto, il frontend, il modello dati e le regole già implementate sono stati mantenuti, mentre le attività svolte da n8n, cioè la normalizzazione e il coordinamento delle fasi successive, sono state spostate nel backend. Questa sezione descrive quindi solo le differenze rispetto alla soluzione originale. Il confronto tra le due versioni, considerando sia i tempi di esecuzione sia gli aspetti non prestazionali, viene presentato nella Sezione 7.3.

6.5.1 Spostamento della normalizzazione nel backend

La normalizzazione era la parte eseguita direttamente nei nodi *Check file* e *Normalizzazione*. Le stesse operazioni sono state raccolte in un nuovo modulo TypeScript, il file `etlNormalization.ts`. Le conversioni rimangono quelle già descritte nella Sezione 6.2.2.

Il *parser*^G TypeScript gestisce anche virgole e ritorni a capo racchiusi tra virgolette, senza dividere il record nel punto sbagliato. I blocchi rimangono da 15.000 righe, ma vengono passati direttamente a `normalizeCsvChunk` e salvati nello staging dal repository, senza richieste HTTP verso n8n.

Il Codice 6.7 mostra una parte di `normalizeCsvChunk`. Per ogni riga viene creato un oggetto con i campi dello staging. Le funzioni di supporto puliscono i valori e li convertono nel tipo previsto, mentre l'offset del blocco permette di mantenere il numero originale della riga.

```
1  const { headers } = validateRequiredHeaders(input.csvText);
2  const rows = parseCsv(
3    input.csvText.replace(/\uFEFF/, ""), ", ";
4  const dataRows = rows.slice(1);
5
6  for (let i = 0; i < dataRows.length; i++) {
7    const rawRow = dataRows[i];
8    const row: Record<string, unknown> = {};
9
10   headers.forEach((header, index) => {
11     if (header) row[header] = rawRow[index] ?? null;
12   });
13
14   output.push({
15     run_id: input.runId,
16     company_id: input.companyId,
17     source_row_number:
18       Number(input.dataRowOffset ?? 0) + i + 2,
19     season_code:
20       cleanString(get(row, "SEASON_CODE"), 100),
21     season_year: cleanInt(get(row, "YEAR")),
22     item_code:
23       cleanString(get(row, "ITEM_CODE"), 100),
24     item_price_acq:
25       cleanDecimal(get(row, "ITEM_PRICE_ACQ")),
26     validation_status: "PENDING",
27   });
28 }
```

Codice 6.7: Estratto della normalizzazione di un blocco CSV

6.5.2 Gestione del flusso dal backend

Il metodo `uploadCsv` prepara la run e avvia `processUploadAsync`. L'elaborazione prosegue in background, così il backend può restituire subito al frontend l'identificativo della run.

Il Codice 6.8 mostra la parte principale del nuovo flusso. I blocchi vengono normalizzati uno alla volta e inseriti nello staging. Terminata questa fase, il backend richiama direttamente il motore di validazione e il metodo che popola le tabelle di frontiera.

```
1  for (const [index, chunk] of chunks.entries()) {
2      const normalized = normalizeCsvChunk({
3          runId,
4          companyId: COMPANY_ID,
5          csvText: chunk.csvText,
6          dataRowOffset: chunk.dataRowOffset,
7      });
8
9      const inserted = await this.stagingRepository
10         .insertNormalizedRows(runId, normalized.rows);
11      totalInserted += inserted;
12  }
13
14  await this.runRepository.setRunStatus(runId, "NORMALIZED");
15  await this.validationEngine.validateRun(runId);
16  await this.populateBorderTables(runId);
```

Codice 6.8: Esecuzione del flusso ETL direttamente dal backend

Per ogni blocco e per ogni fase completata viene registrato un evento della run. Se la normalizzazione fallisce, i dati di staging già creati per quella run vengono eliminati e lo stato passa a **FAILED**. La rivalidazione dopo le correzioni segue lo stesso principio: il backend azzerà gli errori precedenti, esegue di nuovo la validazione e ricrea le tabelle di frontiera senza coinvolgere n8n.

6.5.3 Coda del caricamento definitivo

Durante il confronto, Sinesy ha fatto notare che più caricamenti definitivi potevano essere avviati nello stesso momento. Questa situazione non era gestita nella soluzione originale e avrebbe potuto portare più run a scrivere contemporaneamente in `keepit_svil`. Poiché la modifica è emersa durante la fase successiva del lavoro, per comodità la coda è stata aggiunta soltanto nei branch `main-code-only` e `ai-mapping`.

La coda si trova nel service e mantiene le richieste nell'ordine in cui sono arrivate. In questo modo viene eseguito un solo caricamento definitivo alla volta.

Quando l'Utente avvia il caricamento, `loadRun` controlla che la stessa run non sia già in corso o in attesa. La richiesta viene poi aggiunta a `loadQueue`. Se è già attivo un altro caricamento, la run passa allo stato `LOAD_QUEUED`; in caso contrario viene avviata subito.

Il metodo riportato nel Codice 6.9 estrae una richiesta alla volta e attende la fine di `executeLoad` prima di passare alla successiva. Il blocco `finally` libera sempre la coda anche quando si verifica un errore.

```
1 private async processLoadQueue(): Promise<void> {
2     if (this.processingLoadQueue) return;
3     this.processingLoadQueue = true;
4
5     try {
6         while (this.loadQueue.length > 0) {
7             const job = this.loadQueue.shift();
8             if (!job) break;
9
10            this.currentLoadRunId = job.runId;
11            try {
12                await this.executeLoad(job.runId, job.userId);
13            } catch (ex) {
14                logger.error(ex,
15                    `Errore esecuzione load in coda run ${job.runId}`);
16            } finally {
17                this.currentLoadRunId = null;
18            }
19        }
20    } finally {
21        this.processingLoadQueue = false;
22        if (this.loadQueue.length > 0)
23            void this.processLoadQueue();
24    }
25 }
```

Codice 6.9: Esecuzione in sequenza dei caricamenti definitivi

La coda è mantenuta nella memoria del backend, mentre lo stato `LOAD_QUEUED` viene salvato nel database. Alla prima richiesta ricevuta dopo un riavvio, il service recupera le run rimaste in questo stato e le inserisce nuovamente nella coda.

Prima di iniziare la scrittura viene inoltre ripetuta la validazione completa. Questo controllo serve a rilevare eventuali modifiche avvenute nel frattempo nei dati di riferimento, ad esempio la rimozione di un fornitore o di un codice IVA. Se vengono trovate nuove righe non valide, il caricamento viene annullato e la run torna a essere correggibile.

La Figura 6.9 mostra il messaggio visualizzato quando una run è in attesa. L'Utente non deve eseguire altre operazioni, perché il caricamento parte automaticamente al termine di quello precedente.

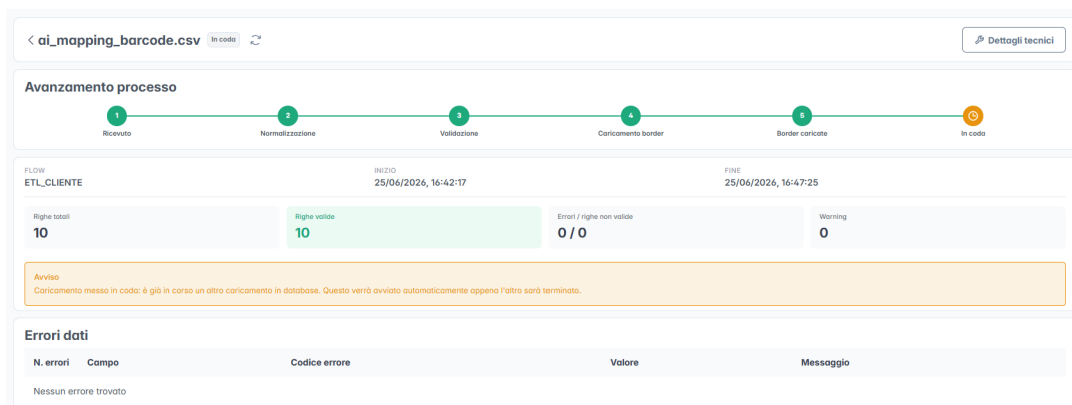


Figura 6.9: Caricamento definitivo in coda nella versione senza n8n

6.6 Implementazione del mapping assistito dall'intelligenza artificiale

Il mapping è stato aggiunto alla versione senza n8n come estensione successiva. Questa funzione permette di elaborare CSV che usano nomi di colonna diversi dal tracciato previsto dal sistema ETL. La progettazione è stata presentata nella Sezione 5.6; di seguito viene descritto come è stata realizzata.

6.6.1 Preparazione della richiesta

Prima di avviare una nuova run, il frontend richiede al backend una preview del mapping. Il backend legge l'installazione e raccoglie fino a cinque righe non vuote da usare come esempio. I valori di esempio permettono di capire quali dati contiene ogni colonna e, di conseguenza, a quale campo del sistema deve essere associata.

I nomi delle colonne vengono anche normalizzati: sono rimossi gli spazi esterni, le lettere vengono convertite in maiuscolo e i separatori diventano caratteri di sottolineatura. Da questo elenco viene calcolato un hash *SHA-256*^G, usato per riconoscere tracciati già elaborati in precedenza. Il Codice 6.10 mostra questa preparazione.

```
1 function normalizeSourceHeaderName(header: string): string {
2     return header
3         .trim()
4         .replace(/\uFEFF/, "")
5         .toUpperCase()
6         .replace(/[~A-Z0-9]+/g, "_")
7         .replace(/~_+|_+$/g, "")
8         .replace(/_+/g, "_");
9 }
10
11 const normalizedSourceHeaders = sourceHeaders
12     .map(normalizeSourceHeaderName);
13 const headerHash = computeHash(
14     JSON.stringify(normalizedSourceHeaders));
15 const sampleRows = dataRecords
16     .filter(record => parseCsvLine(record)
17         .some(value => value.trim() !== ""))
18     .slice(0, 5);
```

Codice 6.10: Preparazione delle intestazioni e delle righe di esempio

Se le intestazioni sono già quelle previste, il CSV può proseguire senza modifiche. In caso contrario, il service cerca nella tabella `brd_etl_column_mappings` un mapping attivo con lo stesso hash. Se non lo trova, viene richiesta una nuova proposta a Gemini.

6.6.2 Generazione e controllo del mapping

La richiesta inviata a Gemini contiene le intestazioni originali, quelle normalizzate, le righe di esempio, l'elenco dei campi ETL disponibili e quello dei campi obbligatori. Nel *prompt*^G viene spiegato anche il significato dei campi che possono essere confusi più facilmente, come `SUPPLIER_CODE`, `ITEM_SUPPLIER` e `MAIN`.

Gemini deve restituire soltanto un oggetto JSON. Per ogni colonna sorgente indica il campo ETL associato, le colonne che non è riuscito a collegare, gli eventuali campi obbligatori mancanti e se il file può essere elaborato. La richiesta usa una temperatura pari a zero, così da rendere la risposta il più possibile stabile, e viene interrotta se supera il tempo massimo configurato.

La risposta dell'intelligenza artificiale non viene applicata direttamente. Il backend controlla che:

- le colonne sorgente indicate esistano davvero nel CSV;
- i campi di destinazione appartengano al tracciato ETL;
- lo stesso campo ETL non sia assegnato a più colonne;

- siano presenti tutti i campi obbligatori;
- i valori di esempio siano compatibili con alcune associazioni più ambigue.

Ad esempio, una colonna contenente soltanto valori booleani non può essere associata a `ITEM_SUPPLIER`, perché potrebbe in realtà rappresentare il campo `MAIN`. Se uno di questi controlli fallisce, l'importazione non prosegue e viene mostrato il motivo dell'errore.

Dopo la conferma, il backend riscrive l'intestazione nel formato canonico e riordina i valori delle righe nella stessa posizione dei campi ETL. Le colonne non associate vengono ignorate, mentre i campi facoltativi non presenti rimangono vuoti. Da questo punto il CSV può essere elaborato dalla stessa normalizzazione usata dalla versione senza `n8n`.

Il mapping confermato viene salvato insieme all'hash delle intestazioni e può essere riutilizzato per i file successivi con la stessa struttura. Solo quando l'Utente conferma un nuovo mapping per lo stesso hash, quello precedente passa allo stato `OBSOLETE`. In questo modo rimane attiva una sola versione del mapping.

6.6.3 Conferma da parte dell'Utente

Quando il CSV è già conforme o viene riconosciuto tramite lo storico, il caricamento prosegue senza mostrare passaggi aggiuntivi. Se invece il mapping è stato generato da Gemini, il frontend apre una finestra di conferma prima di creare la run.

Per ogni colonna del CSV, la finestra di conferma riporta il campo ETL proposto. L'Utente può cambiare l'associazione tramite il menu a scelta e può espandere la riga per vedere fino a tre valori presenti nel file. Nella parte superiore sono indicati i campi obbligatori e lo stato generale del mapping.

Il pulsante di conferma rimane disabilitato se manca un campo obbligatorio oppure se due colonne sono associate allo stesso campo ETL. La Figura 6.10 mostra la finestra completa con un esempio di associazione duplicata.

Quando l'Utente conferma, il frontend invia al backend il mapping eventualmente modificato. Prima di avviare l'importazione, il backend esegue un ultimo controllo come spiegato in precedenza.

Conferma mappatura colonne

Il file CSV contiene colonne non standard. L'intelligenza artificiale ha proposto una prima mappatura tra i campi del tuo file e quelli richiesti dal sistema ETL. Prima di procedere puoi correggere, spostare o completare manualmente le associazioni, anche usando colonne che l'AI aveva inizialmente escluso.

CAMPI OBBLIGATORI PER IL MAPPING

SEASON_CODEYEARBRAND_CODESUPPLIER_CODE

ITEM_SUPPLIERITEM_CODEITEM_DESCRIPTION_IT

SHOP_DESCRIPTION_IT

CONTROLLO MAPPING

Sono presenti associazioni duplicate sugli stessi campi ETL:
SEASON_DESCRIPTION_IT

Colonna nel CSV	Campo ETL	Stato
> COD_STAGIONE	SEASON_DESCRIPTION_IT	Duplicato
> DESC_STAGIONE	SEASON_DESCRIPTION_IT	Duplicato
> STAGIONE	SEASON	Proposto da AI
> ANNO	YEAR	Proposto da AI
> COD_MARCA	BRAND_CODE	Proposto da AI
> DESC_MARCA	BRAND_DESCRIPTION	Proposto da AI
> COD_FORNITORE	SUPPLIER_CODE	Proposto da AI
> ITEM_FORNITORE	ITEM_SUPPLIER	Proposto da AI

[ANNULLA](#)[CONFERMA E AVVIA IMPORTAZIONE](#)

Figura 6.10: Controllo di un'associazione duplicata nel mapping

Capitolo 7

Valutazione della soluzione

Questo capitolo valuta il lavoro svolto durante lo stage. La prima parte confronta i risultati con gli obiettivi e i requisiti iniziali. Vengono poi descritte le prove eseguite e le differenze tra la versione con n8n, quella sviluppata interamente nel backend e l'estensione dedicata al mapping assistito dall'intelligenza artificiale.

7.1 Raggiungimento degli obiettivi e verifica dei requisiti

7.1.1 Raggiungimento degli obiettivi

Gli obiettivi principali sono stati raggiunti. Il lavoro è partito dal confronto tra gli strumenti disponibili ed è arrivato alla realizzazione di un sistema integrato nel boilerplate messo a disposizione da Sinesy. La Tabella 7.1 riassume i risultati ottenuti.

Obiettivo	Esito	Risultato
Analisi e scelta degli strumenti	Raggiunto	Le tecnologie sono state confrontate attraverso lo studio e la realizzazione di Proof of Concept.
Realizzazione del flusso ETL	Raggiunto	Il sistema gestisce caricamento, normalizzazione, validazione, correzione e caricamento definitivo.
Controllo della qualità dei dati	Raggiunto	Gli errori vengono individuati, mostrati e corretti prima della scrittura nelle tabelle definitive.
Tracciabilità delle elaborazioni	Raggiunto	Run, eventi, errori, correzioni e caricamenti vengono registrati e sono consultabili dal frontend.
Gestione di tracciati CSV diversi	Raggiunto	Il mapping assistito permette di associare intestazioni differenti al modello comune.
Gestione di volumi elevati	Raggiunto	L'elaborazione a blocchi ha gestito file con centinaia di migliaia di righe in tempi adeguati.
Integrazione con FourWayShop	Raggiunto nel prototipo	Il progetto usa il boilerplate e le tecnologie aziendali, ma richiede alcuni adattamenti prima dell'utilizzo in produzione.
Gestione multi-tenant	Raggiunto	Run, dati e query sono associati all'azienda, permettendo di separare i dati dei diversi tenant.

Tabella 7.1: Raggiungimento dei principali obiettivi del progetto

7.1.2 Verifica dei requisiti

La Tabella 7.2 riassume i requisiti svolti e quelli previsti, distinguendoli per tipologia e priorità. In ogni cella, il primo numero indica i requisiti svolti e il secondo quelli totali. Il simbolo “–” indica che non erano previsti requisiti per quella combinazione.

Tipologia	Obbligatoria	Desiderabili	Opzionali	Totale
Funzionali	50/50	24/24	0/2	74/76
Di qualità	9/9	7/7	–	16/16
Di vincolo	7/7	–	–	7/7
Totale	66/66	31/31	0/2	97/99

Tabella 7.2: Requisiti svolti rispetto al totale, suddivisi per tipologia e priorità

Gli unici requisiti non svolti sono **RF-OP-01** e **RF-OP-02**, entrambi opzionali. Riguardavano l'avvio programmato delle integrazioni e il supporto di formati diversi dal CSV.

7.2 Modalità di valutazione

Le prove sono state svolte durante lo sviluppo e hanno riguardato il flusso completo. Non sono stati eseguiti benchmark specifici per misurare nel dettaglio tempi di esecuzione e di risposta, latenza e consumo di risorse. Lo scopo principale era verificare il corretto funzionamento del sistema e controllare che l'elaborazione terminasse in un tempo adeguato alle esigenze di Sinesy.

Sono stati provati il caricamento del CSV, la normalizzazione, la validazione, la correzione degli errori, la rivalidazione e il caricamento definitivo. Per ogni fase sono stati controllati lo stato della run, i conteggi mostrati nel frontend e i dati scritti nelle tabelle coinvolte. L'uso di file di dimensioni diverse ha inoltre permesso di verificare l'elaborazione a blocchi con volumi crescenti.

Le versioni con e senza n8n sono state provate con gli stessi file e la stessa configurazione. Il mapping è stato valutato separatamente, usando CSV con intestazioni diverse da quelle previste dal sistema ETL.

7.3 Valutazione e confronto delle versioni realizzate

7.3.1 Correttezza dei risultati

Con gli stessi file, le due versioni hanno prodotto gli stessi conteggi, errori di validazione e dati nelle tabelle di frontiera e definitive. L'utilizzo o meno di n8n non modifica quindi il risultato finale.

7.3.2 Tempi di esecuzione osservati

Nelle prove, un CSV di circa 100.000 righe richiedeva indicativamente due minuti per completare la validazione e il popolamento delle tabelle di frontiera. Il caricamento definitivo poteva richiedere fino ad altri due minuti, in base al numero di inserimenti e aggiornamenti da eseguire. Il tempo complessivo rimaneva quindi inferiore al limite di dieci minuti indicato nei requisiti.

I tempi delle due versioni sono risultati molto simili. Durante le prove non sono stati annotati con precisione, quindi i valori riportati di seguito sono soltanto indicativi. Con circa 100.000 righe, n8n era più veloce di circa dieci secondi,

mentre con 200.000 righe la differenza arrivava a circa venti secondi. Rispetto a un processo che dura alcuni minuti, questa differenza non è stata ritenuta importante nella scelta.

7.3.3 Monitoraggio e gestione degli errori

Per quanto riguarda il monitoraggio, n8n offre un vantaggio chiaro rispetto alla versione senza n8n. La piattaforma dispone già di un sistema che, per ogni esecuzione, mostra l'esito e il nodo in cui il flusso si è interrotto. Per ogni singolo nodo sono inoltre disponibili i dati ricevuti, quelli prodotti e l'eventuale messaggio di errore.

Durante lo sviluppo questa interfaccia ha permesso di individuare rapidamente i problemi nei workflow, come nell'esempio mostrato nella Figura 7.1.

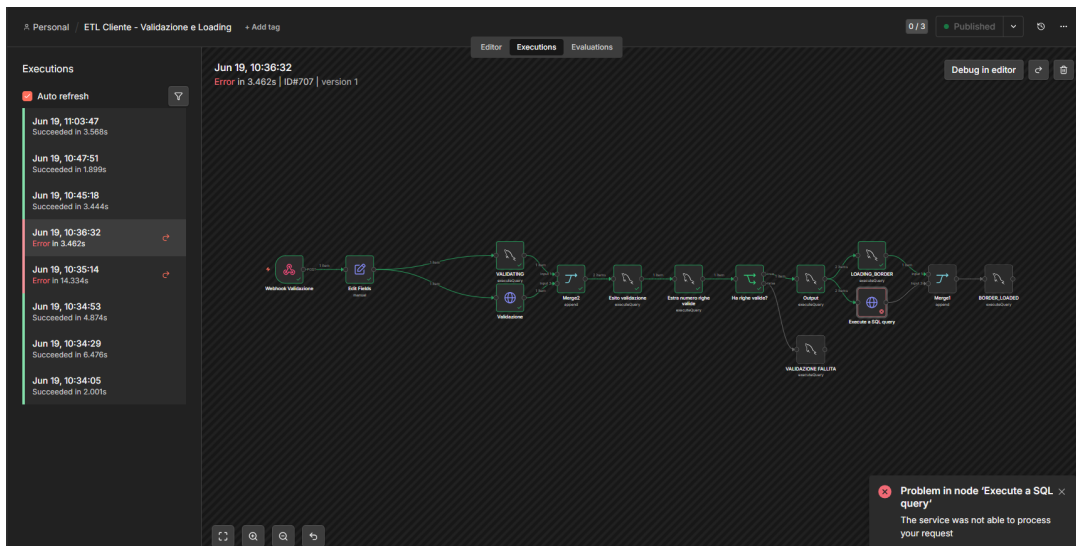


Figura 7.1: Esempio di errore mostrato dall'interfaccia di n8n

Nella versione senza n8n non è disponibile un sistema di log già pronto. Le informazioni consultabili sono soltanto quelle che sono state programmate nel frontend e nel backend oppure salvate nel database. Per ottenere maggiori dettagli sarebbe quindi necessario aggiungere altro codice dedicato.

7.3.4 Manutenibilità e semplicità di gestione

Per quanto riguarda la manutenibilità, le due versioni presentano vantaggi diversi. n8n rende più semplice comprendere e modificare il flusso, mentre la versione backend facilita il lavoro sul codice condiviso. L'interfaccia grafica di n8n mostra attraverso i nodi la sequenza delle operazioni e permette di costruire un workflow senza scrivere tutto da codice. Se le funzioni disponibili non

sono sufficienti, è comunque possibile creare nodi personalizzati, ma in questo caso è necessario programmare. Lo strumento è risultato abbastanza semplice da imparare e consente anche a chi ha meno esperienza con i linguaggi di programmazione di comprendere la struttura generale del processo.

La versione sviluppata soltanto nel backend è meno immediata da leggere, perché per comprenderla è necessario conoscere il codice e la struttura del progetto. In compenso, offre piena libertà sulla logica.

Durante lo stage l'edizione gratuita di n8n è stata gestita senza particolari problemi, anche perché il progetto coinvolgeva soltanto due stagisti. In un team più ampio, come quello di Sinesy, la condivisione dei workflow tramite l'esportazione e l'importazione dei file JSON sarebbe però più complessa e richiederebbe maggiore attenzione per evitare di sovrascrivere le modifiche. L'integrazione diretta con GitHub era disponibile soltanto con piani diversi da quello utilizzato [19, 20]. Nella versione backend, invece, il lavoro di gruppo viene gestito tramite Git come il resto del codice.

n8n richiede inoltre un proprio ambiente Docker e aggiunge un componente da distribuire e mantenere, mentre la versione backend rimane all'interno dell'applicazione predisposta da Sinesy.

7.3.5 Valutazione del mapping assistito dall'intelligenza artificiale

Il mapping assistito è risultato utile per gestire CSV con intestazioni diverse. Gemini propone l'associazione tra le colonne ricevute e i campi del sistema, lasciando all'Utente la possibilità di correggerla prima della conferma. La mappatura può poi essere riutilizzata per file con la stessa struttura, evitando di preparare una normalizzazione specifica per ogni tracciato.

Le prove non hanno coinvolto un numero di file sufficiente per misurare quanto spesso le proposte fossero corrette, ma hanno comunque mostrato che questa funzione riduce il lavoro necessario per adattare un nuovo CSV.

7.3.6 Valutazione complessiva

La Tabella 7.3 riassume le principali differenze emerse.

Aspetto	Versione con n8n	Versione backend
Correttezza	Produce i risultati previsti.	Produce gli stessi risultati.
Tempi	Leggermente più veloce con volumi elevati.	Circa dieci secondi in più ogni 100.000 righe nelle prove svolte.
Comprensione del flusso	Workflow grafico semplice da seguire.	Richiede la lettura del codice TypeScript.
Monitoraggio	Sistema completo e già disponibile.	Limitato alle informazioni programmate nel progetto.
Lavoro in gruppo	Semplice con due sviluppatori, ma più complesso con un team ampio.	Gestito con il normale flusso Git del team.
Flessibilità	Estendibile con nodi personalizzati, che richiedono codice.	Pieno controllo della logica applicativa.
Distribuzione	Richiede anche l'ambiente n8n in Docker.	Rimane nel backend già previsto dal progetto.

Tabella 7.3: Confronto tra la versione con n8n e la versione backend

Nessuna delle due versioni è risultata migliore in ogni aspetto. n8n facilita la costruzione e il controllo del flusso, mentre il backend semplifica il lavoro sul codice condiviso e offre maggiore libertà. Le differenze nei tempi sono troppo piccole per determinare la scelta.

Al termine dello stage Sinesy non aveva ancora scelto quale versione integrare. La decisione potrà quindi essere presa considerando soprattutto le esigenze di gestione e manutenzione.

Capitolo 8

Conclusioni

Questo capitolo presenta i limiti della soluzione e i possibili sviluppi futuri. Vengono poi descritte le conoscenze acquisite durante lo stage e alcune considerazioni personali sull'esperienza e sul lavoro svolto.

8.1 Limiti e possibili miglioramenti

8.1.1 Limiti della soluzione attuale

Il sistema gestisce soltanto file CSV e non accetta altri formati di input. Inoltre, il progetto è un prototipo costruito sul boilerplate di Sinesy: prima dell'utilizzo in produzione, l'azienda dovrà adattare alcune configurazioni e logiche per integrarlo completamente in FourWayShop e provarlo con dati aziendali reali.

8.1.2 Possibili sviluppi futuri

Per valutare meglio le prestazioni, si potrebbe aggiungere un sistema di benchmark in grado di raccogliere dati precisi sui tempi di esecuzione e di risposta, sulla latenza e sul consumo di risorse. Queste informazioni sarebbero utili per entrambe le versioni, soprattutto per quella senza n8n, nella quale dovrebbero essere raccolte attraverso codice dedicato.

La soluzione potrebbe inoltre essere estesa con la programmazione automatica delle integrazioni e il supporto di formati diversi dal CSV. Se Sinesy scegliesse n8n, potrebbe predisporre un ambiente adatto alla gestione di più utenti e tenant e collegare i workflow a Git, rendendo più semplice il lavoro di gruppo.

8.2 Esperienza di stage

8.2.1 Conoscenze acquisite

Lo stage mi ha permesso di conoscere meglio i processi ETL e di comprenderne le principali difficoltà. Lavorare sull'intero flusso ha mostrato i problemi che possono presentarsi durante la normalizzazione, la validazione e il caricamento di dati provenienti da clienti diversi.

Dal punto di vista tecnico, ho imparato a utilizzare n8n e ho approfondito TypeScript, Node.js, React, MySQL, Sequelize e Docker.

Lo stage mi ha aiutato anche a migliorare il confronto con il resto del team, esponendo i problemi incontrati e le scelte effettuate e tenendo conto dei suggerimenti ricevuti. Il progetto è stato svolto insieme a un'altra stagista ed è stato quindi necessario suddividere le attività, coordinare le modifiche e organizzare il lavoro comune.

8.2.2 Considerazioni personali

L'esperienza di tirocinio è stata positiva. Il team ha creato un ambiente sereno, è sempre stato disponibile in caso di dubbi o difficoltà e, allo stesso tempo, ha lasciato libertà nelle scelte e nell'organizzazione del lavoro.

Sono soddisfatto del risultato raggiunto, perché all'inizio dello stage non immaginavo che saremmo riusciti a completare tutti i requisiti obbligatori e quasi tutti quelli definiti. Inoltre la possibilità di lavorare a un problema reale di Sinesy e sapere che quanto realizzato potrà essere riutilizzato dall'azienda ha dato ancora più motivazione durante lo svolgimento del progetto.

8.3 Considerazioni finali

Il progetto ha raggiunto l'obiettivo di rendere più semplice e controllato il processo di importazione dei dati dei clienti di Sinesy, soddisfacendo tutti i requisiti obbligatori. Il confronto tra le due versioni ha inoltre chiarito i rispettivi vantaggi nella gestione del flusso e del codice.

Il mapping assistito rende la soluzione più adatta a CSV con tracciati differenti e riduce il lavoro richiesto per ogni nuovo cliente. Il sistema potrà essere migliorato con gli sviluppi futuri descritti in precedenza, ma Sinesy ha comunque considerato buono il lavoro svolto. Il progetto può quindi essere usato come base per una futura integrazione in FourWayShop.

Sitografia

- [1] Four Way Shop. *Four Way Shop REST API documentation*. URL: <https://developers.fourwayshop.it/> (visitato il giorno 03/08/2026) (cit. a p. 37).
- [2] n8n. *n8n Documentation*. URL: <https://docs.n8n.io> (visitato il giorno 05/08/2026) (cit. alle pp. 39, 42).
- [3] Apache Airflow. *Documentation*. URL: <https://airflow.apache.org/> (visitato il giorno 04/08/2026) (cit. alle pp. 39, 43).
- [4] Apache NiFi. *Apache NiFi User Guide*. URL: <https://nifi.apache.org/nifi-docs/user-guide.html> (visitato il giorno 05/08/2026) (cit. alle pp. 39, 44).
- [5] Mage AI. *Mage Documentation*. URL: <https://docs.mage.ai> (visitato il giorno 07/09/2026) (cit. a p. 39).
- [6] Qlik. *Qlik Automate*. URL: <https://help.qlik.com/en-US/evaluation-guides/Content/data-integration/qlik-app-automation.htm> (visitato il giorno 07/08/2026) (cit. alle pp. 39, 47).
- [7] Google Cloud. *Dataflow overview*. URL: <https://docs.cloud.google.com/dataflow/docs/overview> (visitato il giorno 07/09/2026) (cit. a p. 39).
- [8] Apache Spark. *Documentation*. URL: <https://spark.apache.org/docs/latest/> (visitato il giorno 07/09/2026) (cit. a p. 39).
- [9] Google Cloud. *Cloud Data Fusion overview*. URL: <https://docs.cloud.google.com/data-fusion/docs/concepts/overview> (visitato il giorno 05/08/2026) (cit. alle pp. 39, 46).
- [10] Google Cloud. *Managed Service for Apache Airflow documentation*. URL: <https://docs.cloud.google.com/composer/docs> (visitato il giorno 05/08/2026) (cit. alle pp. 39, 46).

- [11] Google Cloud. *Running services on a schedule*. URL: <https://docs.cloud.google.com/run/docs/triggering/using-scheduler> (visitato il giorno 07/09/2026) (cit. a p. 39).
- [12] Google Cloud. *Workflows overview*. URL: <https://docs.cloud.google.com/workflows/docs/overview> (visitato il giorno 07/09/2026) (cit. a p. 39).
- [13] Apache Beam. *Apache Beam Documentation*. URL: <https://beam.apache.org/documentation/> (visitato il giorno 08/09/2026) (cit. a p. 39).
- [14] Google Cloud. *Use auto data quality*. URL: <https://docs.cloud.google.com/dataplex/docs/use-auto-data-quality> (visitato il giorno 04/08/2026) (cit. a p. 39).
- [15] Prefect. *Prefect Documentation*. URL: <https://docs.prefect.io/> (visitato il giorno 08/09/2026) (cit. a p. 39).
- [16] Dagster. *Dagster Documentation*. URL: <https://docs.dagster.io/> (visitato il giorno 08/09/2026) (cit. a p. 39).
- [17] Airbyte. *Airbyte Documentation*. URL: <https://docs.airbyte.com/> (visitato il giorno 07/09/2026) (cit. a p. 39).
- [18] Meltano. *Meltano Documentation*. URL: <https://docs.meltano.com/> (visitato il giorno 07/09/2026) (cit. a p. 39).
- [19] n8n. *Workflow sharing*. URL: <https://docs.n8n.io/workflows/sharing/> (visitato il giorno 07/09/2026) (cit. a p. 100).
- [20] n8n. *Create environments with source control*. URL: <https://docs.n8n.io/source-control-environments/create-environments/> (visitato il giorno 07/09/2026) (cit. a p. 100).

Acronimi e abbreviazioni

4WS Abbreviazione del nome FourWayShop, la suite software di Sinesy. 1

API Acronimo di *Application Programming Interface*; insieme di regole e operazioni attraverso le quali due sistemi software comunicano. 6

CDAP Acronimo di *Cask Data Application Platform*; piattaforma open source sulla quale si basa Google Cloud Data Fusion. 46

CORS Acronimo di *Cross-Origin Resource Sharing*; meccanismo che regola le richieste web effettuate tra origini differenti. 53

CSS Acronimo di *Cascading Style Sheets*; linguaggio utilizzato per definire l'aspetto delle pagine web. 54

CSV Acronimo di *Comma-Separated Values*; formato testuale utilizzato per rappresentare dati tabellari. 7

DAG Acronimo di *Directed Acyclic Graph*; grafo diretto senza cicli utilizzato per rappresentare attività e dipendenze. 43

ELT Acronimo di *Extract, Load, Transform*; indica il processo nel quale i dati vengono estratti, caricati e successivamente trasformati. 38

ETL Acronimo di *Extract, Transform, Load*; indica il processo di estrazione, trasformazione e caricamento dei dati. 2

FTP Acronimo di *File Transfer Protocol*; protocollo utilizzato per trasferire file tra sistemi collegati in rete. 37

GCP Acronimo di *Google Cloud Platform*; piattaforma cloud di Google utilizzata per distribuire e gestire applicazioni e infrastrutture. 34

HTTP Acronimo di *Hypertext Transfer Protocol*; protocollo utilizzato per lo scambio di dati sul Web. 42

IAM Acronimo di *Identity and Access Management*; insieme di strumenti e regole per gestire identità, ruoli e permessi. 40

JSON Acronimo di *JavaScript Object Notation*; formato testuale usato per rappresentare e scambiare dati strutturati. 49

PoC Acronimo di *Proof of Concept*; realizzazione ridotta usata per verificare la fattibilità di una soluzione. 2

REST Acronimo di *Representational State Transfer*; stile architetturale usato per progettare servizi web. 37

SFTP Acronimo di *SSH File Transfer Protocol*; protocollo utilizzato per trasferire file attraverso una connessione sicura. 41

SHA-256 Acronimo di *Secure Hash Algorithm 256-bit*; funzione crittografica che produce un'impronta di 256 bit. 92

SOAP Acronimo di *Simple Object Access Protocol*; protocollo per lo scambio di messaggi strutturati tra servizi web. 39

SQL Acronimo di *Structured Query Language*; linguaggio utilizzato per interrogare e modificare database relazionali. 53

UML Acronimo di *Unified Modeling Language*; linguaggio standard utilizzato per rappresentare e documentare sistemi software. 9

UUID Acronimo di *Universally Unique Identifier*; identificativo progettato per essere univoco. 61

XLSX Formato basato su Office Open XML utilizzato per i fogli di calcolo. 39

Glossario

Area di staging Area temporanea nella quale vengono conservati i dati durante le fasi di elaborazione, verifica e correzione, prima del caricamento definitivo. 26

Backend Parte di un'applicazione che esegue la logica, accede ai dati ed espone i servizi usati dal frontend. 2

Boilerplate Struttura iniziale di un progetto che contiene configurazioni, tecnologie e componenti già predisposti. 72

Branch Ramo di sviluppo Git che permette di modificare una versione del progetto separatamente dalle altre. 72

Endpoint Indirizzo attraverso il quale è possibile richiamare una specifica operazione di un'API. 58

Firestore Authentication Servizio della piattaforma Firebase utilizzato per gestire l'autenticazione e l'identità degli utenti. 34

Framework Struttura software che fornisce componenti e regole di base per sviluppare un'applicazione. 39

Frontend Parte di un'applicazione che gestisce l'interfaccia e l'interazione con l'Utente. 2

Hash Valore di lunghezza fissa calcolato a partire da un insieme di dati e usato per riconoscerne il contenuto. 66

Legacy Sistema o componente sviluppato in passato e ancora utilizzato. 36

Mapping Associazione tra le colonne di un file sorgente e i campi previsti dal sistema di destinazione. 2

n8n Piattaforma di automazione basata su nodi, utilizzata per progettare e orchestrare flussi di lavoro e integrazioni tra sistemi. 2

Node.js Ambiente di esecuzione multiplatforma che consente di utilizzare JavaScript al di fuori del browser, in particolare per lo sviluppo di applicazioni lato server. 34

Normalizzazione dei dati Trasformazione dei dati in una struttura e in una rappresentazione uniformi, adatte alle successive elaborazioni. 2

Parser Componente che legge un contenuto strutturato e ne riconosce i singoli elementi. 88

Prompt Istruzione fornita a un modello di intelligenza artificiale per indicare il risultato richiesto. 93

React Libreria JavaScript utilizzata per realizzare interfacce utente mediante componenti riutilizzabili. 34

Retail Settore economico relativo alla vendita diretta di beni o servizi al consumatore finale. 1

Run Singola esecuzione di un processo o di un flusso di integrazione, della quale vengono registrati stato, avanzamento ed esito. 6

Self-hosted Software installato e gestito direttamente sulla propria infrastruttura. 42

Tenant Cliente o organizzazione che utilizza uno spazio logicamente isolato all'interno di un sistema condiviso; in un'architettura multi-tenant, dati e risorse di ciascun tenant rimangono separati da quelli degli altri. 2

TypeScript Linguaggio di programmazione basato su JavaScript che introduce un sistema di tipizzazione statica. 34

Webhook Indirizzo web richiamato da un sistema per comunicare un evento o avviare un'operazione in un altro sistema. 51

Wizard Procedura guidata che divide un'attività in passaggi successivi e accompagna l'Utente fino al completamento. 68

Workflow Sequenza organizzata di attività eseguite per completare un processo. 39