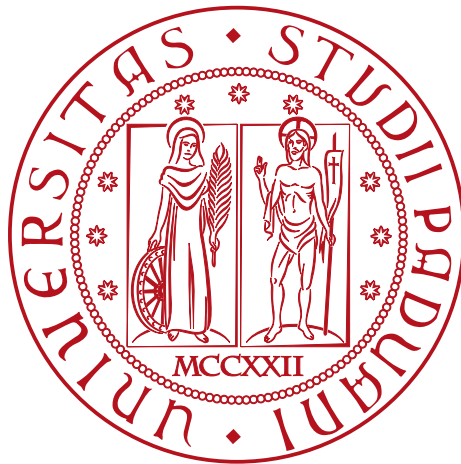


Università degli studi di Padova

DIPARTIMENTO DI MATEMATICA "TULLIO LEVI-CIVITA"

CORSO DI LAUREA IN INFORMATICA



**Integrazione di API e servizi gestionali
per l'automazione delle richieste di
supporto: il caso Vision Ticketing**

Tesi di laurea triennale

Relatore

Prof. Marco Zanella

Laureando

Nicolò Bovo

Matricola 2042885

*«L'automazione non sostituisce l'uomo,
lo libera dalla ripetizione»*

– Nicolò Bovo

Ringraziamenti

Innanzitutto, vorrei esprimere la mia gratitudine al Prof. Marco Zanella, relatore della mia tesi, per la disponibilità immensa, l'interesse mostrato nel seguire il mio lavoro, la gentilezza nel rispondere alle mie domande e l'aiuto fondamentale fornito durante la stesura di questo progetto.

Desidero ringraziare con affetto i miei genitori e i miei nonni per il sostegno costante e per essermi stati vicini in ogni momento durante gli anni di studio. Un pensiero particolare va a mia nonna, che mi ha accompagnato fino a poco tempo fa, e a mio zio, che purtroppo non ci sono più ma restano sempre con noi.

Desidero ringraziare i miei amici per tutti i bellissimi anni passati insieme, con una menzione speciale a Diego per l'aiuto prezioso che mi ha fornito.

Ci tengo infine a ringraziare i colleghi di VISIONEIMPRESA S.r.l. Società Benefit e il tutor aziendale Francesco Turra per avermi dato l'opportunità di lavorare a questo progetto affascinante e per il supporto dimostrato.

Ringrazio infine tutti coloro che non ho citato, e in modo particolare chi c'è stato davvero ed è rimasto anche sveglio con me quando serviva.

Padova, Dicembre 2025

Nicolò Bovo

Sommario

Il presente documento descrive il lavoro svolto durante il periodo di stage curricolare, della durata di circa trecentoventi ore, dal laureando Nicolò Bovo presso l'azienda VISIONEIMPRESA S.r.l. Società Benefit. Lo stage è stato condotto sotto la supervisione del tutor aziendale Francesco Turra, mentre il prof. Prof. Marco Zanella ha ricoperto il ruolo di tutor accademico.

Questa tesi documenta il progetto **Vision Ticketing**, un sistema integrato per l'automazione della gestione delle richieste di assistenza tecnica. Il progetto si articola in tre componenti principali: lo sviluppo di **API SOAP in ambiente .NET** per l'integrazione tra il portale web e il gestionale Vision Enterprise; l'adattamento e personalizzazione di un **portale web di ticketing** preesistente, derivato da Office Group, con modifiche al frontend (utilizzando DDEV e Docker) e personalizzazioni grafiche specifiche per VisionImpresa; la realizzazione di **EmailTicketReader**, un prototipo di servizio automatico che monitora la casella assistenza tramite Microsoft Graph API, identifica i clienti nel database aziendale e genera automaticamente i ticket nel gestionale.

L'obiettivo complessivo è l'automazione dell'intero flusso di gestione delle circa 6.300 richieste annuali di supporto, dalla ricezione via email o portale web alla creazione del ticket in Vision Enterprise, riducendo significativamente l'intervento manuale della reception e i tempi operativi. Il progetto include anche modifiche strutturali al database (creazione della tabella WebReparto, aggiornamento delle tabelle Comm, DocTes2 e ImpostaB2B) e un'ampia fase di testing con strumenti come SoapUI e Postman per garantire la piena operatività del sistema.

Organizzazione del testo

Il **primo capitolo** introduce l'azienda VisioneImpresa, il contesto del progetto Vision Ticketing e le motivazioni personali che hanno portato alla scelta di questo percorso di stage;

Il **secondo capitolo** descrive in dettaglio l'organizzazione dello stage, la durata, il rapporto con il tutor aziendale Francesco Turra, la metodologia di lavoro adottata e l'analisi dei rischi identificati durante la pianificazione;

- Il terzo capitolo** presenta un'analisi approfondita dei requisiti del progetto, suddivisi in obbligatori, desiderabili e opzionali, seguita dall'analisi degli stakeholders coinvolti, dai casi d'uso principali e da una panoramica completa delle funzionalità del sistema;
- Il quarto capitolo** illustra la progettazione e lo sviluppo del sistema di ticketing online, partendo dalle tecnologie adottate (.NET 6, C#, SOAP, SQL Server, DDEV, Docker, DevExpress), proseguendo con l'architettura del sistema, l'implementazione delle sette API SOAP backend, le modifiche strutturali al database Vision Enterprise, lo sviluppo e la personalizzazione del portale web frontend, e conclude con le attività di testing e validazione tramite SoapUI e Postman. Per ogni fase vengono dettagliate le problematiche riscontrate e le soluzioni adottate;
- Il quinto capitolo** documenta la progettazione e lo sviluppo del prototipo EmailTicketReader per l'automazione della lettura delle email, descrivendo l'integrazione con Microsoft Graph API, l'autenticazione OAuth2 tramite Azure Active Directory, la logica di identificazione dei clienti nel database, il parsing del contenuto email, la creazione automatica dei documenti CH in Vision Enterprise, e conclude con le attività di testing specifiche e i limiti del prototipo realizzato;
- Il sesto capitolo** presenta le conclusioni del progetto, analizzando il raggiungimento degli obiettivi prefissati, le competenze tecniche acquisite durante lo stage, le principali difficoltà incontrate nel percorso, i possibili sviluppi futuri del sistema Vision Ticketing e le considerazioni personali sull'esperienza formativa.

Convenzioni tipografiche

Durante la stesura del testo ho scelto di adottare le seguenti convenzioni tipografiche:

- Gli acronimi, le abbreviazioni e i termini di uso non comune menzionati vengono definiti nel glossario, situato alla fine del documento (p. 76);
- Per la prima occorrenza dei termini riportati nel glossario viene utilizzata la seguente nomenclatura: *termines*;
- I termini in lingua straniera non di uso comune o facenti parte del gergo tecnico sono evidenziati con il carattere *corsivo*;
- I nomi di funzioni, classi o variabili appartenenti ad un linguaggio di programmazione vengono scritti con un carattere **monospaziato**;
- I nomi di tabelle del database sono indicati in formato `dbo.NomeTabella`;

- Le citazioni ad un libro o ad una risorsa presente nella bibliografia (p. 80) saranno affiancate dal rispettivo numero identificativo, es. [1];
- I blocchi di codice sono rappresentati nel seguente modo:

Indice

1	Introduzione	1.
1.1	Contesto e motivazione dello stage	1.
1.2	Presentazione dell'azienda	1.
1.3	Il progetto Vision Ticketing	2.
2	Piano di stage	4.
2.1	Organizzazione temporale	4.
2.2	Metodologia di lavoro	5.
2.3	Vincoli e sfide	6.
2.4	Analisi dei rischi	6.
2.5	Ambiente di lavoro	7.
3	Analisi dei requisiti	10.
3.1	Introduzione	10.
3.2	Requisiti del progetto	10.
3.2.1	Requisiti obbligatori	10.
3.2.1.1	RO1: Sviluppo API SOAP backend in C# ...	10.
3.2.1.2	RO2: Sviluppo interfacce con librerie DevExpress	11.
3.2.1.3	RO3: Integrazione con database Vision Enterprise	11.
3.2.2	Requisiti desiderabili	12.
3.2.2.1	RD1: Creazione documentazione tecnica	12.
3.2.2.2	RD2: Piano di test completo	12.
3.2.3	Requisiti opzionali	13.
3.2.3.1	ROP1: Sistema EmailTicketReader	13.
3.2.3.2	ROP2: Integrazione moviTICKET	13.
3.2.3.3	ROP3: Conversione chiamate vocali in testo .	13.
3.3	Requisiti non funzionali	14.
3.3.1	RNF1: Prestazioni	14.
3.3.2	RNF2: Affidabilità	14.
3.3.3	RNF3: Sicurezza	14.
3.3.4	RNF4: Manutenibilità	14.

3.3.5	RNF5: Compatibilità	15.
3.4	Analisi degli stakeholders	15.
3.4.1	ST1: Reception	15.
3.4.2	ST2: Team assistenza tecnica	15.
3.4.3	ST3: Help desk	16.
3.5	Lista dei casi d'uso principali	16.
Area 1.	Autenticazione e accesso	16.
UC1	Login al portale	16.
Area 2.	Gestione ticket	17.
UC2	Creazione nuovo ticket	17.
UC3	Visualizzazione lista ticket	17.
UC4	Visualizzazione dettaglio ticket	18.
UC5	Inserimento sollecito	18.
UC6	Chiusura ticket	19.
4	Sistema di Ticketing	20.
4.1	Introduzione	20.
4.2	Tecnologie adottate	20.
4.2.1	Backend: .NET 6 e C#	20.
4.2.2	Protocollo SOAP	21.
4.2.3	Database: SQL Server 2022	21.
4.2.4	Frontend: DDEV e Docker	21.
4.2.5	Librerie DevExpress	22.
4.3	Architettura del sistema	22.
4.3.1	Flussi di lavoro principali	22.
4.3.1.1	Flusso di creazione ticket	22.
4.3.1.2	Flusso di consultazione ticket	23.
4.3.1.3	Flusso di sollecito	23.
4.3.1.4	Flusso di chiusura	24.
4.4	Implementazione API SOAP backend	24.
4.4.1	Panoramica delle funzioni	25.
4.4.2	Implementazione nf_oit_ins_ticket	27.
4.4.3	Implementazione nf_oit_get_tickets	29.
4.4.4	Implementazione nf_oit_sollecito	31.
4.4.5	Implementazione nf_oit_chiudi segnalazione	32.
4.5	Modifiche al database	33.
4.5.1	Nuova tabella dbo.WebReparto	33.
4.5.2	Modifiche a tabelle esistenti	34.
4.6	Sviluppo e personalizzazione frontend	35.
4.6.1	Configurazione ambiente DDEV	36.
4.6.2	Personalizzazioni grafiche	36.
4.6.3	Modifiche funzionali	37.
4.6.4	Integrazione con API backend	38.

4.7	Testing e validazione	39.
4.7.1	Test funzionali con SoapUI	39.
4.7.2	Test di integrazione con Postman	40.
4.7.3	Test di carico	40.
4.7.4	Problematiche rilevate durante il testing	40.
4.8	Riepilogo delle funzionalità implementate	41.
5	EmailTicketReader	44.
5.1	Introduzione	44.
5.2	Obiettivi del servizio	44.
5.3	Tecnologie utilizzate	45.
5.3.1	Microsoft Graph API	45.
5.3.2	OAuth2 con Azure Active Directory	45.
5.3.3	.NET 6 e librerie di supporto	46.
5.4	Architettura del servizio	46.
5.5	Configurazione Azure Active Directory	47.
5.5.1	Registrazione applicazione	47.
5.5.2	File di configurazione	47.
5.6	Implementazione del servizio	48.
5.6.1	Classe principale EmailTicketReaderService	48.
5.6.2	Acquisizione token OAuth2	51.
5.6.3	Recupero email non lette	53.
5.6.4	Filtraggio email pertinenti	54.
5.6.5	Identificazione cliente nel database	56.
5.6.6	Gestione utente generico per email non identificate ..	58.
5.6.7	Conversione HTML to text	59.
5.6.8	Elaborazione completa di una email	61.
5.6.9	Creazione ticket tramite API esistente	64.
5.6.10	Marcatura email come letta	66.
5.7	Sistema di logging	66.
5.8	Testing del prototipo	69.
5.8.1	Test unitari	69.
5.8.2	Test di integrazione	69.
5.9	Limiti del prototipo	70.
5.9.1	Gestione allegati incompleta	70.
5.9.2	Campi ticket non completamente popolati	71.
5.9.3	Mancanza di gestione thread email	71.
5.9.4	Assenza di interfaccia di monitoraggio	71.
5.9.5	Gestione errori basilare	72.
5.10	Conclusioni sul prototipo	72.
6	Conclusioni	73.
6.1	Raggiungimento degli obiettivi	73.

6.2	Competenze acquisite	74.
6.3	Difficoltà incontrate	74.
6.4	Valutazione personale	75.
Glossario		76.
Bibliografia		80.

Elenco delle Figure

Figura 1	Architettura del sistema Vision Ticketing	2.
Figura 2	Home Page Sistema Vision Ticketing	42.
Figura 3	Creazione di un Ticket	42.
Figura 4	Dettaglio di un Ticket	43.
Figura 5	Tutti i Ticket	43.

Elenco delle Tabelle

Tabella 1	Pianificazione temporale dello stage	5.
Tabella 2	Analisi dei rischi del progetto	7.
Tabella 3	Funzioni API SOAP implementate	26.

Elenco dei Codici Sorgente

Codice 1	Implementazione del metodo nf_oit_ins_ticket pt.1	28.
Codice 2	Implementazione del metodo nf_oit_ins_ticket pt.2	29.
Codice 3	Implementazione del metodo nf_oit_get_tickets	30.
Codice 4	Implementazione del metodo nf_oit_sollecito	32.
Codice 5	Implementazione del metodo nf_oit_chiudi segnalazione	33.
Codice 6	Creazione della tabella WebReparto	34.
Codice 7	Modifica della tabella Comm	34.
Codice 8	Modifica della tabella DocTes2	35.
Codice 9	Inserimento configurazioni in ImpostaB2B	35.
Codice 10	Configurazione Docker-based Development Environment (DDEV) per l'ambiente di sviluppo	36.
Codice 11	Esempio di chiamata SOAP dal frontend	38.

Codice 12	Configurazione Cross-Origin Resource Sharing (CORS) nel backend	39.
Codice 13	File di configurazione appsettings.json	48.
Codice 14	Classe principale EmailTicketReaderService pt1	50.
Codice 15	Classe principale EmailTicketReaderService pt2	51.
Codice 16	Metodo per l'acquisizione del token OAuth2	52.
Codice 17	Metodo per il recupero delle email non lette	53.
Codice 18	Metodo per il filtraggio delle email pertinenti pt.1	55.
Codice 19	Metodo per il filtraggio delle email pertinenti pt.2	56.
Codice 20	Metodo per l'identificazione del cliente tramite email . . .	57.
Codice 21	Gestione fallback con anagrafica generica	58.
Codice 22	Metodo per la conversione da HTML a testo	60.
Codice 23	Metodo per l'elaborazione completa di una email pt.1 . . .	62.
Codice 24	Metodo per l'elaborazione completa di una email pt.2 . . .	63.
Codice 25	Metodo per l'elaborazione completa di una email pt.3 . . .	64.
Codice 26	Metodo per la creazione del ticket tramite API SOAP . .	65.
Codice 27	Metodo per marcare un'email come letta	66.
Codice 28	Implementazione del sistema di logging su file	68.

Capitolo 1

Introduzione

1.1 Contesto e motivazione dello stage

Il presente elaborato documenta l'attività di stage curricolare svolta presso VisioneImpresa s.r.l. Società Benefit. Lo stage si inserisce nel percorso formativo del Corso di Laurea in Informatica dell'Università degli Studi di Padova.

La scelta di questo percorso è stata motivata dall'interesse per le tecnologie di integrazione tra sistemi software e l'automazione dei processi aziendali. Il progetto Vision Ticketing offriva l'opportunità di lavorare con diverse tecnologie: Application Programming Interface (API) SOAP per l'integrazione backend, container Docker per lo sviluppo frontend e Microsoft Graph API per la gestione automatica delle email. Particolarmente stimolante è stata la possibilità di confrontarsi con problematiche concrete di un sistema in produzione che gestisce circa 6.300 richieste di assistenza annuali.

1.2 Presentazione dell'azienda

VisioneImpresa s.r.l. Società Benefit è una software house italiana con sede a Pernumia (PD), specializzata nella progettazione e realizzazione di software gestionali per aziende. Il prodotto principale è *Vision Enterprise*, un Enterprise Resource Planning (ERP) [1] completo e modulare che copre contabilità, magazzino, fatturazione, produzione e gestione del personale.

L'azienda commercializza le proprie soluzioni attraverso due canali:

- **Vendita diretta** a circa 340 aziende (principalmente nel Triveneto), con assistenza di primo livello
- **Rete di rivenditori** con circa 850 licenze attive in tutta Italia, con assistenza di secondo livello

Come Società Benefit, VisioneImpresa persegue finalità di beneficio comune con politiche attente all'equilibrio vita-lavoro, formazione continua e sostenibilità ambientale.

Lo stage è stato seguito dal tutor aziendale Francesco Turra, Amministratore e Responsabile di prodotto, che ha coordinato l'attività garantendo supporto tecnico e organizzativo.

1.3 Il progetto Vision Ticketing

VisioneImpresa fornisce assistenza tecnica continua ai propri clienti per garantire il corretto funzionamento di Vision Enterprise. Le richieste di assistenza possono riguardare:

- Supporto operativo nell'utilizzo del gestionale (registrazione fatture, gestione magazzino, configurazione listini)
- Segnalazioni di malfunzionamenti o comportamenti anomali del software
- Richieste di informazioni su procedure operative e normative fiscali
- Necessità di personalizzazioni per processi aziendali specifici
- Problemi tecnici (connessione database, errori di configurazione)

Attualmente, queste richieste arrivano tramite telefonate o email assistenza. In entrambi i casi, la reception registra manualmente ogni richiesta in Vision Enterprise - il gestionale sviluppato da VisionImpresa e utilizzato anche internamente. La reception accede tramite client desktop alla sezione gestione chiamate e compila un form per creare un documento «CH» (CHiamata).

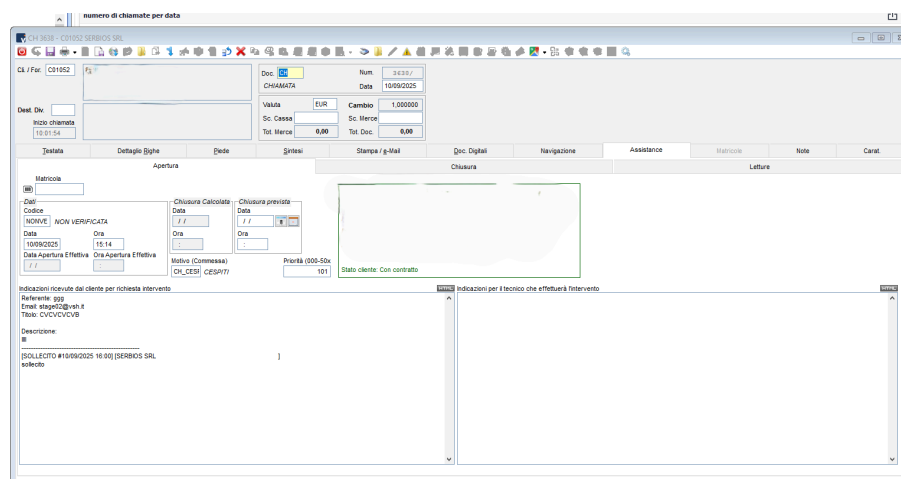


Figura 1: Architettura del sistema Vision Ticketing

I documenti CHiamata (CH) sono record strutturati nel database che formalizzano ogni richiesta, contenendo:

- Dati cliente e referente
- Data/ora apertura, area funzionale (contabilità, magazzino, fatturazione), reparto assegnato.

- Descrizione dettagliata del problema
- Allegati (screenshot, documenti) salvati nella tabella DOCDIG
- Storico progressivo delle attività del team assistenza
- Informazioni di chiusura (data/ora, note finali)

Il flusso di gestione prevede che il documento CH venga verificato dal help desk, preso in carico dal tecnico specializzato, aggiornato progressivamente con le attività svolte, e chiuso alla risoluzione. Ad ogni aggiornamento, Vision Enterprise genera automaticamente un'email di notifica al cliente.

Questo processo manuale presenta criticità quantificabili:

Elevato carico di lavoro: Con 6.300 richieste annuali (circa 25/giorno), ogni inserimento richiede 3-5 minuti, risultando in 2-3 ore giornaliere (25-37% dell'orario lavorativo della reception) dedicate esclusivamente alla registrazione ripetitiva.

Errori di trascrizione: Dati incompleti, selezione errata di area/reparto, allegati mancanti, errori di trascrizione che causano ritardi e necessità di ricontattare il cliente.

Tempi di risposta non ottimali: Tra contatto del cliente e presa in carico passa il tempo di registrazione manuale (3-5 minuti per ticket). Nei picchi (lunedì mattina, post-chiusure) si accumulano 10-15 richieste con ritardi cumulativi di 30-75 minuti.

Accessibilità limitata: I clienti non possono aprire ticket fuori orario (8:40-18:00), non hanno visibilità sullo stato delle richieste, devono sempre passare dalla reception per verifiche, e non dispongono di storico consultabile.

Il progetto Vision Ticketing nasce per automatizzare questo processo con obiettivi quantificabili: ridurre il carico della reception del 60-70% (risparmio 1.5-2 ore/giorno), eliminare gli errori di inserimento, ridurre i tempi di presa in carico di 3-5 minuti per ticket, offrire un canale self-service 24/7 ai clienti, e ottenere dati strutturati per analisi.

Il progetto si articola in sei aree prioritarie:

1. **Portale web di ticketing** per i 340 clienti diretti (apertura ticket, visualizzazione stato, storico, solleciti, chiusure), derivato dal portale Office Information Technologies
2. **API SOAP in C#** come ponte tra portale e database Vision Enterprise, con sette funzioni (creazione ticket, recupero lista con filtri, dettaglio, solleciti, chiusura, recupero reparti/aree programma, attività collegate)
3. **EmailTicketReader** per monitoraggio automatico della casella email e creazione automatica documenti CH tramite Microsoft Graph API e OAuth2
4. **Integrazione moviTICKET** per utilizzare le nuove API eliminando latenze di sincronizzazione

5. **Integrazione VOIP 3CX** per identificazione automatica chiamante e apertura form pre-compilata
 6. **Speech-to-text** per conversione automatica chiamate vocali in testo
- Durante lo stage (8 settimane, 320 ore) sono state affrontate le prime tre aree prioritarie: sviluppo completo delle API SOAP, adattamento del portale web con personalizzazione grafica e funzionale, e prototipo funzionante di EmailTicketReader.

Capitolo 2

Piano di stage

2.1 Organizzazione temporale

Lo stage si è svolto interamente in presenza presso la sede di VisioneImpresa a Pernumia (PD), con una durata complessiva di circa 320 ore distribuite su 8 settimane. L'orario di lavoro concordato prevedeva 40 ore settimanali per le prime sette settimane e 20 ore per l'ultima settimana, seguendo l'orario aziendale standard dalle 8:40 alle 12:40 e dalle 14:00 alle 18:00, dal lunedì al venerdì.

La prima giornata è stata dedicata all'attività di *onboarding*, durante la quale sono stati forniti gli accessi ai sistemi aziendali e le dotazioni hardware necessarie (notebook HP, monitor aggiuntivo, cuffie e telefono VOIP). Sono state inoltre illustrate le procedure operative aziendali e presentati i membri del team con cui avrei collaborato durante il progetto.

Il piano di lavoro iniziale prevedeva la seguente articolazione temporale:

Settimana	Attività	Ore
1	Studio applicativi e database	40
2-3	Sviluppo portale e API Simple Object Access Protocol (SOAP)	80
4-5	Integrazione email (EmailTicketReader)	80
6	Integrazione moviTICKET	40
7	Test e documentazione	40
8	Completamento e consegna	20

Tabella 1: Pianificazione temporale dello stage

2.2 Metodologia di lavoro

La metodologia di lavoro adottata si è ispirata ai principi dello sviluppo agile [2], con una suddivisione del progetto in macro-fasi successive e incrementali. Il rapporto con il tutor aziendale era caratterizzato da incontri settimanali di pianificazione e verifica, solitamente il lunedì mattina, durante i quali venivano discussi gli avanzamenti della settimana precedente, eventuali problematiche riscontrate e gli obiettivi da raggiungere. Oltre agli incontri programmati, erano disponibili confronti estemporanei per chiarimenti tecnici o dubbi sulle specifiche del progetto. Ogni settimana seguiva un ciclo iterativo strutturato in cinque fasi:

1. **Pianificazione:** definizione degli obiettivi settimanali
 2. **Studio e analisi:** comprensione dei requisiti e delle tecnologie necessarie
 3. **Sviluppo:** implementazione delle funzionalità pianificate
 4. **Test:** verifica del corretto funzionamento con SoapUI e Postman
 5. **Revisione:** confronto sui risultati ottenuti e pianificazione successiva
- Durante lo sviluppo, sono state coinvolte diverse figure aziendali per supportare specifici aspetti tecnici: Dario Minella (Direzione sviluppo) per questioni relative all'architettura del sistema e alle best practices in .NET, e membri del Reparto Assistenza per comprendere i flussi operativi del sistema di ticketing esistente e le necessità degli utenti finali.

Gli strumenti di lavoro principali sono stati:

- **Visual Studio 2022** per lo sviluppo delle API SOAP in C Sharp
- **Azure DevOps [3]** per il versionamento del codice sorgente
- **DDEV [4] e Docker [5]** per l'ambiente di sviluppo frontend
- **SQL Server Management Studio** per le modifiche al database
- **SoapUI [6] e Postman [7]** per il testing delle API

La documentazione tecnica è stata redatta progressivamente interagendo costantemente con il team.

2.3 Vincoli e sfide

Durante lo stage sono emersi vincoli significativi che hanno influenzato le modalità di sviluppo:

Vincoli tecnici:

- Compatibilità con il database esistente di Vision Enterprise, modificabile solo tramite aggiunta di nuove tabelle o campi
- Mantenimento della compatibilità con il portale Office Group, richiedendo API con input e output identici
- Necessità di testare in ambiente controllato per non compromettere l'operatività quotidiana

Vincoli temporali:

- Disponibilità limitata del tutor in alcuni periodi per impegni commerciali
- Tempi ristretti per completare le tre macro-aree prioritarie del progetto

Sfide tecnologiche:

- Apprendimento di SOAP, .NET 6 e Microsoft Graph API con OAuth2, tecnologie non approfondite nel percorso universitario
- Comprensione della logica di business complessa del gestionale Vision Enterprise
- Mappatura delle differenze strutturali tra i database di Office Group e Vision Enterprise

2.4 Analisi dei rischi

In fase di pianificazione sono stati identificati e valutati i seguenti rischi:

ID	Descrizione	Probabilità	Strategia di mitigazione
R1	Ritardi nello sviluppo delle API	Media	Priorità chiara sulle funzioni core; sviluppo incrementale
R2	Problemi di integrazione con Vision ERP	Alta	Testing incrementale dopo ogni modifica; ambiente di test dedicato
R3	Complessità OAuth2 per EmailTicketReader	Alta	Studio preliminare della documentazione Microsoft; esempi di codice
R4	Incompatibilità tra portali Office Group e Vision	Bassa	Analisi approfondita delle API esistenti prima dello sviluppo
R5	Impossibilità di completare tutti gli obiettivi	Alta	Focus sulle prime 3 aree prioritarie; obiettivi graduati

Tabella 2: Analisi dei rischi del progetto

Il rischio R5 si è effettivamente materializzato: l'integrazione con **movi-TICKET** (punto 4 del piano originale) non è stata completata nei tempi previsti. Tuttavia, grazie alla chiara definizione delle priorità, le tre aree principali (portale web, API SOAP e prototipo **EmailTicketReader**) sono state portate a termine al 100% per le prime due e ad un 70% il sistema mail, garantendo il raggiungimento degli obiettivi core del progetto. Anche il rischio R2 ha richiesto attenzione costante: le differenze tra la struttura dati di **Office Group** e **Vision Enterprise** hanno richiesto numerose iterazioni di test per garantire la corretta integrazione.

2.5 Ambiente di lavoro

VisioneImpresa ha messo a disposizione un ambiente di lavoro professionale con:

- Postazione dedicata con PC connesso alla rete locale aziendale
- Accesso ai server di test (srvAppVsh01) per frontend e backend
- Credenziali Office365 (stage02) per lo sviluppo e test
- Accesso al database VISIONVI su SQL Server (77.89.26.156SQLEXPRESS2022,1522)
- Repository GIT su Azure DevOps per il versionamento del codice

Capitolo 3

Analisi dei requisiti

3.1 Introduzione

Il presente capitolo descrive in dettaglio l'analisi dei requisiti condotta per il progetto Vision Ticketing. L'analisi è stata svolta durante la prima settimana di stage in collaborazione con il tutor aziendale e ha previsto lo studio del sistema esistente, l'identificazione degli stakeholders coinvolti, la definizione dei requisiti funzionali e non funzionali, e la modellazione dei casi d'uso principali.

L'obiettivo dell'analisi era comprendere a fondo le esigenze dell'azienda e tradurle in specifiche tecniche implementabili, garantendo al contempo la compatibilità con i sistemi esistenti e la scalabilità futura della soluzione.

3.2 Requisiti del progetto

I requisiti sono stati classificati in tre categorie secondo le priorità aziendali: obbligatori, desiderabili e opzionali.

3.2.1 Requisiti obbligatori

I requisiti obbligatori rappresentano le funzionalità essenziali che devono essere implementate per considerare il progetto concluso con successo.

3.2.1.1 RO1: Sviluppo API SOAP backend in C#

Descrizione: Implementare un insieme di API SOAP [8] in ambiente .NET 6 [9] che replichino le funzionalità del sistema Office Group, mantenendo identici input e output per garantire la compatibilità con il portale web esistente.

Dettaglio tecnico: Le API devono essere sviluppate in C# [10] utilizzando il framework .NET 6 e devono interfacciarsi con il database SQL Server [11] di Vision Enterprise. Ogni API deve restituire risposte in formato XML conforme allo standard SOAP 1.2.

Criteri di accettazione:

- Implementazione di almeno 7 funzioni SOAP principali

- Compatibilità totale con le chiamate del portale Office Group
- Gestione corretta degli errori con codici e messaggi standard
- Documentazione completa di ogni endpoint

Funzioni da implementare:

- `nf_oit_ins_ticket`: creazione nuovo ticket
- `nf_oit_get_tickets`: recupero lista ticket con filtri
- `nf_oit_sollecito`: inserimento sollecito su ticket aperto
- `nf_oit_chiudi segnalazione`: chiusura ticket
- `nf_oit_get_sett_personale`: recupero reparti disponibili
- `nf_oit_get_area_programma`: recupero aree programma
- `nf_oit_get_attivita_collegate`: recupero attività collegate a un ticket

3.2.1.2 RO2: Sviluppo interfacce con librerie DevExpress

Descrizione: Adattare e personalizzare il portale web esistente (derivato da Office Group) utilizzando le librerie DevExpress per garantire un'interfaccia professionale e coerente con l'identità visiva di VisioneImpresa.

Dettaglio tecnico: Il portale deve essere configurato utilizzando DDEV e Docker Container Platform (Docker) per l'ambiente di sviluppo, garantendo portabilità e facilità di deployment.

Criteri di accettazione:

- Rimozione di tutte le funzionalità non pertinenti (area interventi, campi non utilizzati)
- Personalizzazione grafica completa (logo, colori, favicon)
- Funzionalità di login, creazione ticket, lista ticket, dettaglio ticket operative
- Gestione corretta degli allegati in fase di creazione ticket
- Integrazione completa con le API backend sviluppate

Modifiche richieste rispetto a Office Group:

- Rimozione area interventi dalla navbar
- Rimozione campi: destinazione, area azienda, priorità (dal form di creazione)
- Rimozione filtro area azienda dalla lista ticket
- Aggiunta campi: CH (codice chiamata), Tecnico (nella lista ticket)
- Rimozione limite 100 caratteri nel campo descrizione
- Rimozione funzione «Reply ticket».

3.2.1.3 RO3: Integrazione con database Vision Enterprise

Descrizione: Garantire l'integrazione completa tra il portale web e il database SQL Server di Vision Enterprise, creando e modificando le tabelle necessarie per supportare le nuove funzionalità.

Dettaglio tecnico: L'integrazione deve avvenire tramite le API SOAP sviluppate, che agiranno da layer intermedio tra frontend e database. Devono essere create nuove tabelle e modificate quelle esistenti secondo le specifiche.

Criteri di accettazione:

- Creazione tabella `dbo.WebReparto` con tutti i campi specificati
- Aggiunta campo `eCommerce` (BIT) alla tabella `dbo.Comm`
- Aggiunta campo `Id_WebReparto` (INT, FK) alla tabella `dbo.DocTes2`
- Aggiunta righe di configurazione in `dbo.ImpostaB2B` (CodDocCH, CodApertCH, CodChiusCH)
- Salvataggio corretto dei documenti CH con tutti i campi popolati
- Gestione degli allegati tramite tabella `DOCDIG`

3.2.2 Requisiti desiderabili

I requisiti desiderabili rappresentano funzionalità che migliorano significativamente la qualità del progetto ma non sono strettamente necessarie per il suo funzionamento base.

3.2.2.1 RD1: Creazione documentazione tecnica

Descrizione: Produrre una documentazione tecnica completa che descriva l'architettura del sistema, le API sviluppate, le modifiche al database e le procedure di deployment.

Contenuti richiesti:

- Specifica tecnica del sistema Vision Ticketing
- Documentazione di ogni API SOAP (input, output, logica)
- Diagrammi UML di architettura e sequenza
- Manuale di installazione e configurazione
- Guida alle modifiche del database

Criteri di accettazione:

- Documentazione salvata su Confluence aziendale
- Diagrammi chiari e aggiornati
- Esempi di chiamate SOAP con SoapUI/Postman
- Procedure di troubleshooting per problemi comuni

3.2.2.2 RD2: Piano di test completo

Descrizione: Definire e documentare un piano di test strutturato che copra tutte le funzionalità implementate, con particolare attenzione ai test funzionali e di integrazione.

Tipologie di test richieste:

- Test funzionali delle singole API con SoapUI/Postman

- Test di integrazione frontend-backend
- Test di compatibilità con Vision Enterprise
- Test di gestione errori e casi limite

Criteri di accettazione:

- Documentazione di alcuni test
- Evidenze dei test eseguiti (screenshot, log)
- Copertura di tutti i flussi principali (creazione, modifica, chiusura ticket)
- Report finale con esiti e problematiche rilevate

3.2.3 Requisiti opzionali

I requisiti opzionali rappresentano estensioni future del sistema che possono essere implementate in fasi successive del progetto.

3.2.3.1 ROP1: Sistema EmailTicketReader

Descrizione: Sviluppare un servizio automatico che monitora la casella email di assistenzas, identifica i clienti nel database e crea automaticamente i ticket in Vision Enterprise.

Nota: Questo requisito è stato parzialmente implementato come prototipo durante lo stage. La versione completa richiederà ulteriori sviluppi.

Funzionalità richieste:

- Connessione a Microsoft 365 tramite Microsoft Graph API
- Autenticazione OAuth2 con Azure Active Directory
- Lettura automatica delle email non lette
- Parsing del contenuto e identificazione mittente
- Creazione automatica documento CH in Vision Enterprise
- Gestione allegati email

3.2.3.2 ROP2: Integrazione moviTICKET

Descrizione: Modificare l'app moviTICKET per utilizzare le nuove API SOAP sviluppate, eliminando le latenze di sincronizzazione attualmente presenti con moviDAT Connector.

Nota: Questo requisito non è stato implementato durante lo stage per priorità di tempo.

3.2.3.3 ROP3: Conversione chiamate vocali in testo

Descrizione: Implementare l'integrazione con il sistema telefonico VOIP 3CX per convertire automaticamente le chiamate in ingresso in testo e creare documenti CH.

Nota: Questo requisito non è stato implementato durante lo stage per priorità di tempo.

3.3 Requisiti non funzionali

Oltre ai requisiti funzionali, sono stati identificati requisiti non funzionali che definiscono le caratteristiche qualitative del sistema.

3.3.1 RNF1: Prestazioni

Descrizione: Il sistema deve garantire tempi di risposta accettabili per non rallentare il flusso di lavoro degli operatori.

Metriche:

- Creazione di un ticket: < 3 secondi
- Recupero lista ticket (fino a 100 elementi): < 2 secondi
- Chiamata API singola: < 1 secondo

3.3.2 RNF2: Affidabilità

Descrizione: Il sistema deve essere stabile e gestire correttamente situazioni di errore senza compromettere l'integrità dei dati.

Requisiti:

- Gestione transazionale delle operazioni sul database
- Logging completo di tutte le operazioni critiche
- Rollback automatico in caso di errori durante l'inserimento
- Retry automatici per errori di rete temporanei

3.3.3 RNF3: Sicurezza

Descrizione: Il sistema deve proteggere i dati sensibili e prevenire accessi non autorizzati.

Misure di sicurezza:

- Autenticazione obbligatoria per l'accesso al portale
- Utilizzo di connessioni HTTPS per tutte le comunicazioni
- Token di autenticazione con scadenza temporale
- Client Credentials Flow (Client Credentials) [12] per EmailTicketReader

3.3.4 RNF4: Manutenibilità

Descrizione: Il codice deve essere ben strutturato, documentato e facilmente manutenibile.

Pratiche adottate:

- Architettura modulare con separazione delle responsabilità
- Dependency Injection per la gestione delle dipendenze
- Commenti nel codice per le sezioni complesse
- Naming convention standard per variabili e funzioni

3.3.5 RNF5: Compatibilità

Descrizione: Il sistema deve essere compatibile con l'infrastruttura tecnologica esistente di VisioneImpresa.

Requisiti di compatibilità:

- SQL Server 2022 come database
- .NET 6 Runtime per il backend
- Browser moderni per il frontend (Chrome, Firefox, Edge, Safari)
- Windows Server 2016+ per il deployment
- Integrazione con Office365 per EmailTicketReader

3.4 Analisi degli stakeholders

Gli stakeholders sono le persone o i gruppi che hanno un interesse nel progetto o che ne sono influenzati. Per il progetto Vision Ticketing sono stati identificati i seguenti stakeholders principali.

3.4.1 ST1: Reception

Ruolo: Attualmente gestisce manualmente tutte le richieste di assistenza in arrivo via telefono o email, registrandole in Vision Enterprise.

Esigenze:

- Riduzione del carico di lavoro manuale
- Riduzione degli errori di inserimento dati

Benefici attesi:

- Automazione della registrazione delle email (EmailTicketReader)
- Riduzione del tempo dedicato all'inserimento manuale
- Possibilità di dedicare più tempo ad attività a valore aggiunto

3.4.2 ST2: Team assistenza tecnica

Ruolo: Consulenti tecnici che gestiscono e risolvono i ticket di assistenza, suddivisi per le due aree di competenza: Vision e Movidat.

Esigenze:

- Accesso rapido alle informazioni del ticket
- Visibilità completa dello storico delle comunicazioni
- Possibilità di aggiornare lo stato e aggiungere note
- Notifiche automatiche via email sui ticket assegnati

Benefici attesi:

- Informazioni più complete e strutturate nei ticket
- Riduzione del tempo di ricerca delle informazioni
- Migliore tracciabilità delle attività svolte

3.4.3 ST3: Help desk

Ruolo: Verifica il contenuto dei ticket creati dalla reception, conferma o modifica la priorità ed eventualmente assegna la richiesta ad altri membri del reparto assistenza.

Esigenze:

- Accesso immediato ai nuovi ticket
- Possibilità di modificare priorità e assegnazione
- Visibilità sul carico di lavoro del team

Benefici attesi:

- Ticket già strutturati e completi di informazioni essenziali
- Riduzione del tempo di verifica iniziale
- Migliore distribuzione del carico di lavoro

3.5 Lista dei casi d'uso principali**Area 1. Autenticazione e accesso****UC1 Login al portale**

Descrizione: Come cliente diretto, voglio poter effettuare il login al portale per accedere alle funzionalità di gestione ticket.

Precondizioni:

- L'utente dispone di credenziali valide (username e password)
- Il portale è accessibile e funzionante

Flusso principale:

1. L'utente accede alla pagina di login del portale
2. L'utente inserisce username e password
3. L'utente clicca sul pulsante «Accedi»
4. Il sistema verifica le credenziali nel database Vision Enterprise
5. Il sistema genera un token di sessione
6. Il sistema reindirizza l'utente all'Area Anagrafica

Postcondizioni:

- L'utente è autenticato e può accedere alle funzionalità del portale
- Il token di sessione è attivo per le richieste successive

Area 2. Gestione ticket

UC2 Creazione nuovo ticket

Descrizione: Come cliente diretto autenticato, voglio poter creare un nuovo ticket di assistenza per segnalare un problema o richiedere supporto.

Precondizioni:

- L'utente ha effettuato il login al portale

Flusso principale:

1. L'utente accede all'Area Ticket tramite la barra laterale
2. L'utente clicca sul pulsante «Aggiungi ticket»
3. L'utente compila i campi obbligatori:
 - Referente
 - Area programma
 - Reparto
 - Titolo
 - Descrizione
4. L'utente eventualmente allega file tramite drag & drop
5. L'utente clicca su «Salva Ticket»
6. Il sistema valida i dati inseriti
7. Il sistema crea un documento CH in Vision Enterprise tramite API
8. Il sistema mostra un messaggio di conferma con il codice ticket

Postcondizioni:

- Un nuovo documento CH è creato nel database Vision Enterprise
- Il ticket è visibile nella lista ticket dell'utente
- Gli allegati sono salvati nella tabella DOCDIG
- Il team assistenza riceve notifica del nuovo ticket

UC3 Visualizzazione lista ticket

Descrizione: Come cliente diretto autenticato, voglio poter visualizzare la lista dei miei ticket per monitorare lo stato delle mie richieste di assistenza.

Precondizioni:

- L'utente ha effettuato il login al portale

Flusso principale:

1. L'utente accede all'Area Ticket
2. L'utente clicca su «Lista ticket»
3. L'utente imposta eventuali filtri di ricerca:
 - Reparto
 - Intervallo date apertura

- Intervallo date chiusura
 - Stato (Aperti/Chiusi)
4. L'utente clicca sul pulsante «Ricerca»
 5. Il sistema recupera i ticket tramite API
 6. Il sistema visualizza i ticket in ordine cronologico decrescente
 7. Per ogni ticket vengono mostrati: codice CH, titolo, stato, data apertura, reparto, tecnico assegnato

Postcondizioni:

- La lista dei ticket è visualizzata secondo i filtri applicati
- L'utente può accedere al dettaglio di ogni ticket
- L'utente può esportare ogni ticket in PDF

UC4 Visualizzazione dettaglio ticket

Descrizione: Come cliente diretto autenticato, voglio poter visualizzare tutti i dettagli di un ticket per consultare lo storico completo delle attività.

Precondizioni:

- L'utente ha visualizzato la lista ticket

Flusso principale:

1. L'utente clicca sul pulsante «Dettaglio» di un ticket
2. Il sistema recupera tutte le informazioni del ticket tramite API
3. Il sistema mostra:
 - Tutti i dati del ticket (referente, titolo, descrizione, stato, date)
 - Elenco degli allegati
 - PDF del ticket
 - Storico delle attività collegate
 - Pulsanti azione (se ticket aperto): «Sollecita» e «Chiudi come risolto»

Postcondizioni:

- Il dettaglio completo del ticket è visualizzato
- L'utente può effettuare azioni sul ticket (sollecito o chiusura)

UC5 Inserimento sollecito

Descrizione: Come cliente diretto autenticato, voglio poter inserire un sollecito su un ticket aperto per richiedere aggiornamenti al team assistenza.

Precondizioni:

- L'utente ha visualizzato il dettaglio di un ticket aperto

Flusso principale:

1. L'utente clicca sul pulsante «Sollecita»
2. Il sistema mostra una finestra di dialogo
3. L'utente inserisce le note del sollecito
4. L'utente clicca su «Conferma e inserisci sollecito»
5. Il sistema registra il sollecito tramite API
6. Il sistema incrementa la priorità del ticket
7. Il sistema aggiunge timestamp e note al ticket
8. Il sistema mostra un messaggio di conferma

Postcondizioni:

- Il sollecito è registrato nel sistema
- La priorità del ticket è incrementata di 1
- Il team assistenza riceve notifica del sollecito

UC6 Chiusura ticket

Descrizione: Come cliente diretto autenticato, voglio poter chiudere un ticket aperto quando il problema è stato risolto.

Precondizioni:

- L'utente ha visualizzato il dettaglio di un ticket aperto
- Il problema è stato risolto

Flusso principale:

1. L'utente clicca sul pulsante «Chiudi come risolto»
2. Il sistema mostra una finestra di dialogo
3. L'utente inserisce eventuali note di chiusura
4. L'utente clicca su «Conferma e chiudi ticket»
5. Il sistema aggiorna il ticket tramite API impostando:
 - Stato = «Chiuso»
 - Data/ora chiusura = timestamp corrente
6. Il sistema mostra un messaggio di conferma
7. I pulsanti «Sollecita» e «Chiudi come risolto» vengono nascosti

Postcondizioni:

- Il ticket è chiuso nel sistema
- Il ticket non è più modificabile
- Il ticket appare nei filtri «Chiusi»

Capitolo 4

Sistema di Ticketing

4.1 Introduzione

Il presente capitolo documenta la progettazione e lo sviluppo del sistema di ticketing online, che rappresenta il cuore del progetto Vision Ticketing. Il sistema è composto da tre componenti principali: le API SOAP backend che gestiscono la logica di business, il database SQL Server che memorizza i dati, e il portale web frontend che fornisce l'interfaccia utente ai clienti. Per ogni componente vengono descritte le tecnologie adottate, le scelte architetturali, l'implementazione effettiva, le problematiche riscontrate e le soluzioni adottate.

4.2 Tecnologie adottate

4.2.1 Backend: .NET 6 e C#

Per lo sviluppo delle API backend è stato scelto il framework **.NET 6** [9] con linguaggio **C#**. La scelta è motivata principalmente dalla compatibilità con l'ecosistema esistente: VisioneImpresa utilizza tecnologie Microsoft per tutto il proprio stack applicativo, Vision Enterprise è sviluppato in ambiente .NET, e l'infrastruttura server è basata su Windows Server. L'adozione di .NET 6 garantisce piena integrazione senza necessità di bridge o adattatori.

.NET 6 è inoltre una versione LTS (Long-Term Support) con supporto garantito a lungo termine [9], requisito fondamentale per applicazioni aziendali. Offre miglioramenti significativi in termini di prestazioni rispetto alle versioni precedenti, con ottimizzazioni del garbage collector e del JIT compiler. Il linguaggio C# fornisce tipizzazione forte, gestione delle eccezioni robusta e strumenti di debugging avanzati.

Tra le alternative valutate, Java con Spring Boot è stato escluso per mancanza di competenze interne e incompatibilità con l'ecosistema Microsoft, .NET Framework 4.8 è stato scartato perché legacy, e Node.js per la necessità di mantenere coerenza con lo stack tecnologico aziendale.

4.2.2 Protocollo SOAP

Per la comunicazione tra frontend e backend è stato adottato il protocollo **SOAP (Simple Object Access Protocol)** [8] anziché REST [13]. Questa scelta, apparentemente controintuitiva in un contesto moderno, è dettata da un requisito specifico: il portale web deriva dal sistema Office Information Technologies che utilizza chiamate SOAP. Mantenere lo stesso protocollo permette di riutilizzare il frontend con modifiche minime, riducendo drasticamente i tempi di sviluppo.

SOAP utilizza inoltre WSDL (Web Services Description Language) per definire contratti formali tra client e server, garantendo che input e output siano sempre conformi alle specifiche. Questo riduce gli errori di integrazione e facilita la validazione automatica. Le caratteristiche tecniche implementate includono: formato messaggi XML con envelope SOAP 1.2, binding HTTP/HTTPS, encoding Document/Literal, e gestione errori tramite SOAP Fault standardizzati.

REST con JSON sarebbe stato più moderno e leggero, ma avrebbe richiesto la riscrittura completa del frontend. GraphQL risultava troppo complesso per le esigenze attuali, mentre gRPC, pur offrendo prestazioni superiori, era incompatibile con il frontend esistente.

4.2.3 Database: SQL Server 2022

Il database utilizzato è **Microsoft SQL Server 2022** [11], già in uso presso VisioneImpresa per Vision Enterprise. La scelta è obbligata dalla necessità di integrarsi con il sistema esistente. Le caratteristiche sfruttate includono transazioni Atomicity, Consistency, Isolation, Durability (ACID) per garantire atomicità e consistenza dei dati, stored procedures per operazioni complesse, e foreign keys per mantenere l'integrità referenziale tra le tabelle.

4.2.4 Frontend: DDEV e Docker

L'ambiente di sviluppo frontend utilizza **DDEV** [4], strumento open-source basato su **Docker** [5] per la gestione di ambienti containerizzati. Docker garantisce che l'ambiente di sviluppo sia identico su qualsiasi macchina, eliminando problemi di portabilità. I container isolano completamente l'applicazione dal sistema host, evitando conflitti. DDEV astrae la complessità di Docker fornendo comandi semplici per gestire l'ambiente, e il file di configurazione `.ddev/config.yaml` è versionabile insieme al codice.

Le alternative come XAMPP/WAMP non erano containerizzate e presentavano problemi di portabilità, Vagrant richiedeva virtualizzazione completa più pesante, e l'ambiente nativo comportava rischi di conflitti.

4.2.5 Librerie DevExpress

Per l'interfaccia utente sono state utilizzate le librerie **DevExpress**, suite commerciale di componenti UI. DevExpress fornisce griglie dati avanzate, form builder e controlli UI già ottimizzati, riducendo i tempi di sviluppo. Il portale Office Group utilizza già DevExpress, permettendo coerenza visiva con minime modifiche. Le funzionalità avanzate includono supporto nativo per filtri, ordinamento, paginazione, drag & drop per upload file e validazione form lato client.

4.3 Architettura del sistema

Il sistema è stato progettato seguendo un'architettura a tre livelli (*three-tier architecture*) [14]:

- **Livello di presentazione (Frontend)**: Portale web accessibile ai clienti tramite browser
- **Livello logico (Backend)**: API SOAP che implementano la logica di business
- **Livello dati (Database)**: SQL Server con il database Vision Enterprise

Questa separazione garantisce modularità, manutenibilità del codice e possibilità di evolvere singoli componenti senza impattare l'intero sistema. L'architettura backend utilizza il pattern nativo di .NET 6 per la gestione delle dipendenze, garantendo testabilità, accoppiamento lasco tra componenti e facile configurabilità.

4.3.1 Flussi di lavoro principali

Prima di analizzare nel dettaglio l'implementazione delle singole funzioni API, è utile comprendere i flussi di lavoro che il sistema deve supportare. Questi flussi corrispondono direttamente ai casi d'uso identificati nella Sezione 3 e rappresentano le interazioni tra utente, frontend, backend e database.

4.3.1.1 Flusso di creazione ticket

Questo flusso implementa il caso d'uso UC2 «Creazione nuovo ticket» descritto nel Capitolo 3.

Il processo di creazione di un nuovo ticket segue questo percorso:

1. L'utente compila il form nel portale web inserendo i dati obbligatori (referente, area programma, reparto, titolo, descrizione) ed eventualmente allega file

2. Il frontend valida i dati lato client e costruisce un messaggio SOAP XML
 3. La chiamata SOAP viene inviata all'API backend `nf_oit_ins_ticket`
 4. Il backend valida nuovamente i dati ricevuti
 5. Il backend recupera le configurazioni di sistema dalla tabella `ImpostaB2B`
 6. Viene generato un codice univoco per il documento CH
 7. Inizia una transazione database
 8. Il backend inserisce il record nella tabella `DocTes` (documento CH)
 9. Se presenti allegati, vengono salvati nella tabella `DOCDIG` con riferimento al documento
 10. La transazione viene confermata (commit)
 11. Il backend restituisce al frontend il codice ticket generato
 12. Il frontend mostra all'utente un messaggio di conferma
- Questo flusso garantisce che la creazione del ticket sia atomica: o tutte le operazioni hanno successo, o nessuna viene applicata (rollback in caso di errore).

4.3.1.2 Flusso di consultazione ticket

Questo flusso implementa i casi d'uso UC3 «Visualizzazione lista ticket» e UC4 «Visualizzazione dettaglio ticket» descritti nel Capitolo 3.

Per visualizzare i propri ticket, l'utente segue questo percorso:

1. L'utente accede alla sezione «Lista ticket» del portale
2. L'utente imposta eventuali filtri (reparto, date, stato)
3. Il frontend chiama l'API `nf_oit_get_tickets` passando i filtri impostati
4. Il backend costruisce dinamicamente una query SQL includendo solo i filtri specificati
5. La query viene eseguita sul database, recuperando i record dalla tabella `DocTes` con join su `WebReparto`
6. I risultati vengono serializzati in formato SOAP XML
7. Il frontend riceve i dati e popola la griglia
8. L'utente può cliccare su «Dettaglio» per un ticket specifico
9. Il frontend chiama `nf_oit_get_attivita_collegate` per recuperare lo storico completo
10. Vengono mostrati tutti i dettagli del ticket incluse le attività svolte

4.3.1.3 Flusso di sollecito

Questo flusso implementa il caso d'uso UC5 «Inserimento sollecito» descritto nel Capitolo 3.

Quando un utente vuole sollecitare un ticket aperto:

1. L'utente visualizza il dettaglio di un ticket aperto
2. Clicca sul pulsante «Sollecita»
3. Inserisce note opzionali in una finestra di dialogo
4. Il frontend chiama l'API `nf_oit_sollecito`
5. Il backend verifica che il ticket sia nello stato «Aperto»
6. Incrementa il campo `Priorita` di 1 unità
7. Concatena le note con timestamp al campo `NoteConcatenate`
8. Aggiorna il record nel database
9. Il sistema Vision Enterprise genera automaticamente un'email al team assistenza
10. Il frontend aggiorna la visualizzazione mostrando la priorità aumentata

4.3.1.4 Flusso di chiusura

Questo flusso implementa il caso d'uso UC6 «Chiusura ticket» descritto nel Capitolo 3.

Per chiudere un ticket risolto:

1. L'utente visualizza il dettaglio di un ticket aperto
2. Clicca sul pulsante «Chiudi come risolto»
3. Inserisce note di chiusura opzionali
4. Il frontend chiama l'API `nf_oit_chiudi segnalazione`
5. Il backend verifica che il ticket sia nello stato «Aperto»
6. Imposta il campo `Stato` al valore configurato per «Chiuso»
7. Registra `DataChiusura` con il timestamp corrente
8. Salva le `NoteChiusura` se fornite
9. Aggiorna il record nel database
10. Il frontend aggiorna la visualizzazione nascondendo i pulsanti di azione

Questi flussi di lavoro guidano l'implementazione delle API SOAP, che vengono ora descritte nel dettaglio.

4.4 Implementazione API SOAP backend

Compreso il contesto architetturale e i flussi di lavoro supportati, passiamo all'analisi dell'implementazione concreta delle API SOAP. Ogni API corrisponde a una specifica operazione richiesta dai casi d'uso e implementa uno dei flussi descritti precedentemente. Le sette funzioni principali coprono tutte le operazioni necessarie per la gestione completa del ciclo di vita di un ticket, dall'apertura alla chiusura.

4.4.1 Panoramica delle funzioni

Sono state implementate sette funzioni SOAP principali, ciascuna con input e output compatibili con il sistema Office Group:

Funzione	Descrizione	Parametri principali
nf_oit_ins_ticket	Creazione nuovo ticket	ID cliente, referente, area programma, reparto, titolo, descrizione, allegati
nf_oit_get_tickets	Recupero lista ticket con filtri	ID cliente, reparto, date apertura/chiusura, stato
nf_oit_sollecito	Inserimento sollecito su ticket aperto	ID ticket, note sollecito
nf_oit_chiudi segnalazione	Chiusura ticket	ID ticket, note chiusura
nf_oit_get_sett_personale	Recupero reparti disponibili	Nessuno
nf_oit_get_area_programma	Recupero aree programma	Nessuno
nf_oit_get_attivita_collegate	Recupero attività collegate a un ticket	ID ticket

Tabella 3: Funzioni API SOAP implementate

4.4.2 Implementazione nf_oit_ins_ticket

La funzione di creazione ticket implementa il Sezione 4.3.1.1 ed è la più complessa del sistema. Il flusso di esecuzione prevede:

1. **Validazione input:** Verifica che tutti i campi obbligatori siano presenti e validi
2. **Recupero configurazioni:** Lettura da `dbo.ImpostaB2B` dei codici documento, stato apertura e chiusura
3. **Generazione codice univoco:** Creazione del codice progressivo per il documento CH
4. **Inserimento documento:** Creazione del record nella tabella `dbo.DocTes` con tutti i campi popolati
5. **Gestione allegati:** Per ogni file allegato, salvataggio nella tabella `dbo.DOCDIG` con collegamento al documento CH
6. **Commit transazione:** Conferma di tutte le operazioni in modo atomico

```

1  [WebMethod]
2  public InstTicketResponse
   nf_oit_ins_ticket(InstTicketRequest request)
3  {
4      using var transaction =
       _connection.BeginTransaction();
5      try
6      {
7          // Validazione
8          ValidateRequest(request);
9
10         // Recupero configurazioni
11         var config = GetImpostaB2B();
12
13         // Generazione codice
14         var codiceUnivoco =
            GenerateCodiceCH(config.CodDocCH);
15
16         // Inserimento documento CH
17         var idDoc = InsertDocumentoCH(request,
            codiceUnivoco, config);
18
19         // Gestione allegati
20         foreach (var allegato in request.Allegati)
21         {
22             InsertAllegato(idDoc, allegato);
23         }
24
25         transaction.Commit();
26         return new InstTicketResponse {
27             Success = true,
28             CodiceTicket = codiceUnivoco
29         };
30     }

```

Codice 1: Implementazione del metodo nf_oit_ins_ticket pt.1

```
1 catch (Exception ex)
2 {
3     transaction.Rollback();
4     return new InsTicketResponse {
5         Success = false,
6         ErrorMessage = ex.Message
7     };
8 }
9 }
```

Codice 2: Implementazione del metodo nf_oit_ins_ticket pt.2

Problematica riscontrata: Durante i primi test, l'allegato non si salvava correttamente nel filesystem.

Soluzione adottata: Sblocco dei permessi nella cartella dove venivano salvati i documenti, garantendo al processo .NET i diritti di scrittura necessari.

4.4.3 Implementazione nf_oit_get_tickets

La funzione di recupero lista ticket implementa il Sezione 4.3.1.2 con filtri dinamici basati sui parametri ricevuti:

```

1  [WebMethod] C#
2  public GetTicketsResponse
3  nf_oit_get_tickets(GetTicketsRequest request)
4  {
5      var query = @"
6          SELECT dt.ID_DocTes, dt.CodiceUnivoco,
7          dt.Titolo, dt.Stato,
8          dt.DataApertura, dt.DataChiusura,
9          wr.DesWebReparto, dt.Tecnico
10         FROM dbo.DocTes dt
11         LEFT JOIN dbo.WebReparto wr ON
12         dt.Id_WebReparto = wr.Id_WebReparto
13         WHERE dt.ID_CliFor = @IdCliente AND
14         dt.TipoDoc = 'CH'";
15
16     if (request.IdReparto.HasValue)
17         query += " AND dt.Id_WebReparto =
18         @IdReparto";
19
20     if (request.DataAperturaDa.HasValue)
21         query += " AND dt.DataApertura >=
22         @DataAperturaDa";
23
24     if (request.DataAperturaA.HasValue)
25         query += " AND dt.DataApertura <=
26         @DataAperturaA";
27
28     if (!string.IsNullOrEmpty(request.Stato))
29         query += " AND dt.Stato = @Stato";
30
31     query += " ORDER BY dt.DataApertura DESC";
32
33     // Esecuzione query e mapping risultati
34     return ExecuteQuery<GetTicketsResponse>(query,
35     parameters);
36 }

```

Codice 3: Implementazione del metodo nf_oit_get_tickets

Problematica riscontrata: Le query con molti filtri attivi risultavano lente (> 5 secondi) su dataset di grandi dimensioni.

Soluzione adottata: Creazione di indici composti sulle colonne più frequentemente filtrate (**ID_CliFor**, **DataApertura**, **Stato**, **Id_WebReparto**), riducendo i tempi di risposta a < 1 secondo.

4.4.4 Implementazione **nf_oit_sollecito**

La funzione di sollecito implementa il Sezione 4.3.1.3 con logica di incremento priorità e concatenazione note:

```

1  [WebMethod]
2  public SollecitoResponse
   nf_oit_sollecito(SollecitoRequest request)
3  {
4      var timestamp = DateTime.Now.ToString("dd/MM/
   yyyy HH:mm:ss");
5      var nuoveNote = $"[SOLLECITO {timestamp}]
   {request.Note}";
6
7      var query = @"
8          UPDATE dbo.DocTes
9          SET Priorita = Priorita + 1,
10             NoteConcatenate =
   CONCAT(NoteConcatenate, @NuoveNote)
11          WHERE ID_DocTes = @IdTicket AND Stato =
   'Aperto'";
12
13     var rowsAffected = ExecuteNonQuery(query,
   parameters);
14
15     if (rowsAffected == 0)
16         return new SollecitoResponse {
17             Success = false,
18             ErrorMessage = "Ticket non trovato o
   già chiuso"
19         };
20
21     return new SollecitoResponse { Success =
   true };
22 }

```

Codice 4: Implementazione del metodo nf_oit_sollecito

4.4.5 Implementazione nf_oit_chiudi_segnaazione

La funzione di chiusura implementa il Sezione 4.3.1.4, aggiornando lo stato e registrando data/ora:

```

1  [WebMethod]
2  public ChiudiResponse
   nf_oit_chiudi segnalazione(ChiudiRequest request)
3  {
4      var query = @"
5          UPDATE dbo.DocTes
6          SET Stato = @CodChiusCH,
7              DataChiusura = @DataChiusura,
8              NoteChiusura = @NoteChiusura
9          WHERE ID_DocTes = @IdTicket AND Stato =
            @CodApertCH";
10
11     // Esecuzione con gestione transazionale
12 }

```

Codice 5: Implementazione del metodo nf_oit_chiudi segnalazione

Problematica riscontrata: Alcuni ticket venivano chiusi senza che le note di chiusura fossero salvate correttamente a causa di caratteri speciali nel testo.

Soluzione adottata: Implementazione di sanitizzazione input con escape dei caratteri speciali SQL e validazione lunghezza massima del campo note.


4.5 Modifiche al database

Le API SOAP descritte nella sezione precedente interagiscono con il database SQL Server di Vision Enterprise. Per supportare le nuove funzionalità del sistema di ticketing online, sono state necessarie alcune modifiche strutturali al database esistente. Queste modifiche includono la creazione di una nuova tabella per i reparti web, l'aggiunta di campi a tabelle esistenti per collegare i ticket ai reparti, e l'inserimento di configurazioni di sistema. Tutte le modifiche sono state progettate per integrarsi armoniosamente con lo schema esistente, evitando di impattare le funzionalità già operative di Vision Enterprise.

4.5.1 Nuova tabella dbo.WebReparto

Per gestire i reparti visibili nel portale web è stata creata una nuova tabella. Questa tabella contiene la lista dei reparti dell'assistenza tecnica (VisionENTERPRISE, moviDAT, ecc.) che possono essere selezionati

durante la creazione di un ticket. La separazione in una tabella dedicata permette di gestire dinamicamente l'elenco dei reparti senza modificare codice:


```
1 CREATE TABLE dbo.WebReparto ( 
2     Id_WebReparto INT PRIMARY KEY IDENTITY(1,1),
3     DesWebReparto NVARCHAR(100) NOT NULL,
4     Email NVARCHAR(255) NULL,
5     Attivo BIT DEFAULT 1,
6     DataCreazione DATETIME DEFAULT GETDATE()
7 );
8
9 -- Popolazione iniziale
10 INSERT INTO dbo.WebReparto (DesWebReparto, Email)
11 VALUES
12 ('VisionENTERPRISE', 'assistenza@vsh.it'),
13 ('moviDAT', 'movidat@vsh.it');
```

Codice 6: Creazione della tabella WebReparto

4.5.2 Modifiche a tabelle esistenti

Per collegare i nuovi dati ai documenti CH esistenti e permettere la selezione delle aree programma visibili nel portale, sono state apportate modifiche non invasive a tre tabelle già presenti nel database:

Tabella dbo.Comm: Questa tabella contiene l'elenco delle aree programma (Contabilità, Magazzino, Fatturazione, ecc.). È stato aggiunto un campo booleano per indicare quali aree devono essere visibili nel portale web [15]:

```
1 ALTER TABLE dbo.Comm ADD eCommerce BIT 
2     DEFAULT 0;
3
4 -- Attivazione per aree programma visibili nel
5 -- portale
6 UPDATE dbo.Comm SET eCommerce = 1
7 WHERE DesComm IN ('Contabilità', 'Magazzino',
8 'Fatturazione', 'Vendite');
```

Codice 7: Modifica della tabella Comm

Tabella dbo.DocTes2: Questa è la tabella principale che contiene tutti i documenti di Vision Enterprise, inclusi i documenti CH (ticket). È stato aggiunto un campo per collegare ogni ticket al reparto web assegnato:

```
1 ALTER TABLE dbo.DocTes2 ADD Id_WebReparto INT
  NULL;
2
3 ALTER TABLE dbo.DocTes2
4 ADD CONSTRAINT FK_DocTes2_WebReparto
5 FOREIGN KEY (Id_WebReparto) REFERENCES
  dbo.WebReparto(Id_WebReparto);
```

Codice 8: Modifica della tabella DocTes2

Tabella dbo.ImpostaB2B: Questa tabella contiene le configurazioni di sistema. Sono state aggiunte tre configurazioni per definire i codici utilizzati dal sistema di ticketing:

```
1 INSERT INTO dbo.ImpostaB2B (Chiave, Valore,
  Descrizione) VALUES
2 ('CodDocCH', 'CH', 'Codice tipo documento
  Chiamata'),
3 ('CodApertCH', 'APERTO', 'Codice stato apertura
  ticket'),
4 ('CodChiusCH', 'CHIUSO', 'Codice stato chiusura
  ticket');
```

Codice 9: Inserimento configurazioni in ImpostaB2B

Problematica riscontrata: L'aggiunta del campo `Id_WebReparto` come foreign key causava errori sui documenti CH esistenti che non avevano questo valore popolato.

Soluzione adottata: Definizione del campo come nullable (`NULL`) per permettere la coesistenza di documenti vecchi (senza reparto web) e nuovi (con reparto web assegnato).

4.6 Sviluppo e personalizzazione frontend

Il frontend del sistema di ticketing è basato sul portale web già sviluppato da Office Group, personalizzato per riflettere l'identità e le esigenze specifiche di VisioneImpresa. Lo sviluppo è avvenuto in ambiente containerizzato per garantire replicabilità e isolamento. In questa sezione

vengono descritte la configurazione dell'ambiente di sviluppo, le personalizzazioni grafiche apportate, le modifiche funzionali richieste dalle specifiche, e l'integrazione con le API backend tramite chiamate SOAP.

4.6.1 Configurazione ambiente DDEV

Per lo sviluppo del frontend è stato utilizzato DDEV, che permette di creare un ambiente di sviluppo containerizzato con Docker. Questo approccio garantisce che ogni sviluppatore lavori esattamente con le stesse versioni di PHP, Apache e MariaDB, eliminando i classici problemi di «funziona sulla mia macchina». L'ambiente è stato configurato con il seguente file `.ddev/config.yaml`:

```
1  name: vision-ticketing
2  type: php
3  php_version: "8.1"
4  webserver_type: apache-fpm
5  router_http_port: "80"
6  router_https_port: "443"
7  xdebug_enabled: false
8  additional_hostnames: []
9  additional_fqdns: []
10 database:
11   type: mariadb
12   version: "10.4"
13 use_dns_when_possible: true
14 composer_version: "2"
15 web_environment: []
```

Codice 10: Configurazione DDEV per l'ambiente di sviluppo

I comandi principali utilizzati durante lo sviluppo:

- `ddev start`: Avvio dell'ambiente containerizzato
- `ddev stop`: Arresto dei container
- `ddev ssh`: Accesso shell al container web per operazioni manuali
- `ddev logs`: Visualizzazione log per debugging

4.6.2 Personalizzazioni grafiche

Il portale è stato personalizzato per riflettere l'identità visiva di VisioneImpresa, sostituendo gli elementi grafici di Office Group:

- **Logo:** Sostituzione del logo Office Group con quello VisioneImpresa nell'header del portale
- **Palette colori:** Adozione dei colori aziendali (blu primario #0066CC, grigio secondario #666666) in pulsanti, header e elementi di interfaccia
- **Favicon:** Sostituzione dell'icona del browser con quella VisioneImpresa
- **Footer:** Aggiornamento con informazioni di contatto VisioneImpresa

4.6.3 Modifiche funzionali

Secondo le specifiche concordate con il tutor aziendale e in linea con il requisito RO2 del Sezione 3, sono state apportate modifiche funzionali per adattare il portale alle esigenze specifiche di VisioneImpresa, rimuovendo elementi non necessari e aggiungendo informazioni rilevanti:

Rimozioni dalla navbar:

- Area Interventi (funzionalità presente in Office Group ma non pertinente per i clienti diretti VisioneImpresa)

Rimozioni dal form di creazione ticket:

- Campo «Destinazione» (non utilizzato nel flusso VisioneImpresa)
- Campo «Area Azienda» (poteva confondere con Area Programma)
- Campo «Priorità» (viene assegnata automaticamente dal sistema, non dal cliente)

Rimozioni dalla lista ticket:

- Filtro «Area Azienda» (coerentemente con la rimozione dal form)

Rimozioni dal dettaglio ticket:

- Funzione «Reply ticket» (non prevista nel flusso operativo VisioneImpresa)
- Possibilità di aggiungere allegati dopo la creazione (gli allegati vanno inseriti solo in fase di apertura ticket)

Aggiunte alla lista ticket:

- Colonna «CH» mostrante il codice univoco della chiamata
- Colonna «Tecnico» mostrante il nome del tecnico assegnato al ticket

Altre modifiche:

- Rimozione del limite di 100 caratteri nel campo descrizione, permettendo testi più lunghi
- Aggiunta di validazione campi obbligatori con messaggi di errore chiari

Problematica riscontrata: La rimozione di alcuni campi dal form causava errori JavaScript perché il codice esistente tentava di accedere a elementi Document Object Model (DOM) non più presenti.

Soluzione adottata: Refactoring del codice JavaScript con controlli di esistenza degli elementi prima dell'accesso (`if (element !== null)`), e rimozione delle funzioni non più necessarie per evitare dead code.

4.6.4 Integrazione con API backend

Il frontend comunica con il backend tramite chiamate SOAP, implementando i flussi descritti nelle sezioni precedenti. Ogni operazione utente (creazione ticket, visualizzazione lista, sollecito, chiusura) genera una chiamata SOAP verso il corrispondente endpoint. Il codice JavaScript è stato configurato per puntare al server API di VisioneImpresa:

```
1  var soapRequest = { JS JavaScript
2      url: "https://api.vsh.it/VisionTicketing.asmx",
3      method: "POST",
4      contentType: "text/xml; charset=utf-8",
5      data: buildSoapEnvelope("nf_oit_ins_ticket",
6          ticketData),
7      success: function(response) {
8          var result = parseSoapResponse(response);
9          if (result.Success) {
10             showSuccessMessage("Ticket creato: " +
11                 result.CodiceTicket);
12             refreshTicketList();
13         } else {
14             showErrorMessage(result.ErrorMessage);
15         }
16     },
17     error: function(xhr, status, error) {
18         showErrorMessage("Errore di comunicazione
19             con il server");
20     }
21 };
```

Codice 11: Esempio di chiamata SOAP dal frontend

Problematica riscontrata: Le chiamate SOAP cross-origin venivano bloccate dal browser per politiche [16].

Soluzione adottata: Configurazione del backend per includere gli header CORS appropriati, permettendo al portale di comunicare con l'API:

```

1 app.UseCors(builder => builder
2     .WithOrigins("https://ticket.vsh.it")
3     .AllowAnyMethod()
4     .AllowAnyHeader());

```

Codice 12: Configurazione CORS nel backend

4.7 Testing e validazione

Una volta completate le implementazioni, il sistema è stato sottoposto a testing sistematico per verificare il corretto funzionamento di tutte le componenti e la loro integrazione. Il testing è stato organizzato su tre livelli: test funzionali delle singole API con SoapUI, test di integrazione end-to-end con Postman, e test di carico per verificare le prestazioni. Ogni test è stato documentato con i risultati attesi e ottenuti.

4.7.1 Test funzionali con SoapUI

SoapUI è stato utilizzato per testare sistematicamente ogni funzione API in isolamento. Per ogni funzione è stata creata una test suite con diversi casi di test che coprono sia i flussi normali che i casi limite ed errore:

Test `nf_oit_ins_ticket`:

- TC1: Creazione ticket con tutti i campi obbligatori → Atteso: Success = true
- TC2: Creazione ticket senza titolo → Atteso: Errore validazione
- TC3: Creazione ticket con allegato → Atteso: Success = true, allegato salvato in DOCDIG
- TC4: Creazione ticket con cliente inesistente → Atteso: Errore cliente non trovato

Test `nf_oit_get_tickets`:

- TC5: Recupero tutti i ticket di un cliente → Atteso: Lista completa ordinata per data
- TC6: Filtro per stato «Aperto» → Atteso: Solo ticket aperti
- TC7: Filtro per intervallo date → Atteso: Solo ticket nell'intervallo
- TC8: Cliente senza ticket → Atteso: Lista vuota

Test `nf_oit_sollecito`:

- TC9: Sollecito su ticket aperto → Atteso: Priorità incrementata, note aggiunte
- TC10: Sollecito su ticket chiuso → Atteso: Errore ticket non modificabile
- TC11: Sollecito con note vuote → Atteso: Success = true, solo timestamp registrato

Test nf_oit_chiudi segnalazione:

- TC12: Chiusura ticket aperto → Atteso: Stato = Chiuso, data chiusura popolata
- TC13: Chiusura ticket già chiuso → Atteso: Errore ticket già chiuso
- TC14: Chiusura con note speciali (caratteri accentati) → Atteso: Success = true, note salvate correttamente

4.7.2 Test di integrazione con Postman

Postman è stato utilizzato per test end-to-end che simulano il flusso completo di un utente reale, verificando l'integrazione tra tutte le componenti del sistema:

Flusso completo testato:

1. Login utente → Ottenimento token sessione
2. Visualizzazione anagrafica → Verifica dati corretti
3. Recupero aree programma → Lista popolata
4. Recupero reparti → Lista popolata
5. Creazione ticket con allegato → Codice ticket restituito
6. Recupero lista ticket → Nuovo ticket presente
7. Visualizzazione dettaglio → Tutti i dati corretti
8. Inserimento sollecito → Priorità aumentata
9. Chiusura ticket → Stato aggiornato
10. Verifica finale → Ticket nei filtri «Chiusi»

Tutti i test sono stati documentati con screenshot delle richieste SOAP e delle risposte ricevute, con evidenze salvate su Confluence per tracciabilità.

4.7.3 Test di carico

Sono stati eseguiti test di carico basilari per verificare che le prestazioni rispettassero i requisiti non funzionali definiti nel Sezione 3:

- Creazione ticket: < 3 secondi (obiettivo raggiunto: media 2.1 secondi)
- Recupero lista 100 ticket: < 2 secondi (obiettivo raggiunto: media 1.4 secondi)
- Chiamata API singola: < 1 secondo (obiettivo raggiunto: media 0.6 secondi)

4.7.4 Problematiche rilevate durante il testing

Problema 1: Timeout su operazioni lunghe

Durante la creazione di ticket con allegati di grandi dimensioni (> 5MB), la chiamata andava in timeout prima del completamento dell'operazione.

Soluzione: Aumento del timeout lato server da 30 a 120 secondi e implementazione di upload asincrono per file di grandi dimensioni, con progress bar per feedback visivo all'utente.

Problema 2: Caratteri speciali nei messaggi SOAP

Alcuni caratteri speciali (< > & “ «) presenti nelle descrizioni ticket causavano errori di parsing XML perché non correttamente escaped.

Soluzione: Encoding XML corretto di tutti i campi testo con funzione di escape dedicata che converte i caratteri speciali nelle rispettive entità XML (<, >, &, ', ").

Problema 3: Concorrenza nella generazione codici

Test simultanei di creazione ticket causavano occasionalmente la generazione di codici duplicati, violando il vincolo di unicità.

Soluzione: Implementazione di lock database tramite `sp_getapplock` per serializzare la generazione dei codici, garantendo unicità anche sotto carico concorrente.

4.8 Riepilogo delle funzionalità implementate

Al termine dello sviluppo, il sistema di ticketing online offre le seguenti funzionalità complete:

Per i clienti diretti:

- Login sicuro con credenziali personali
- Visualizzazione dati anagrafici aziendali
- Creazione ticket con allegati multipli
- Lista ticket con filtri avanzati (reparto, date, stato)
- Dettaglio ticket con storico attività
- Inserimento solleciti su ticket aperti
- Chiusura autonoma ticket risolti
- Esportazione ticket in PDF

Per il sistema VisioneImpresa:

- Integrazione completa con database Vision Enterprise
- Creazione automatica documenti CH
- Gestione allegati in DOCDIG
- Tracciamento completo delle operazioni
- Compatibilità con flussi esistenti del team assistenza

Il sistema è stato validato attraverso testing approfondito e risulta pronto per il deployment in ambiente di produzione, in attesa della fase di formazione degli utenti che lo useranno.

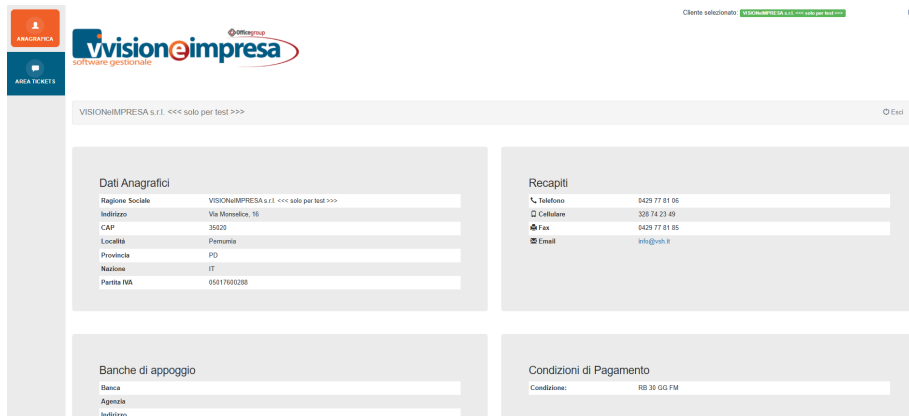


Figura 2: Home Page Sistema Vision Ticketing

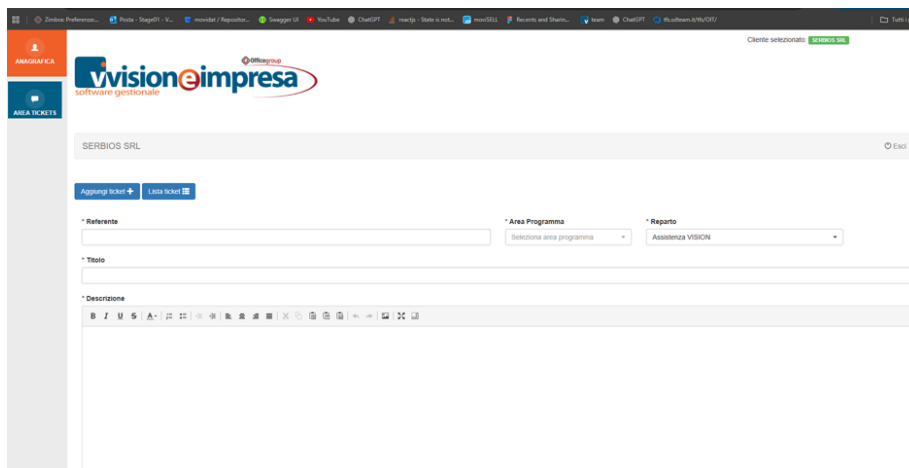


Figura 3: Creazione di un Ticket

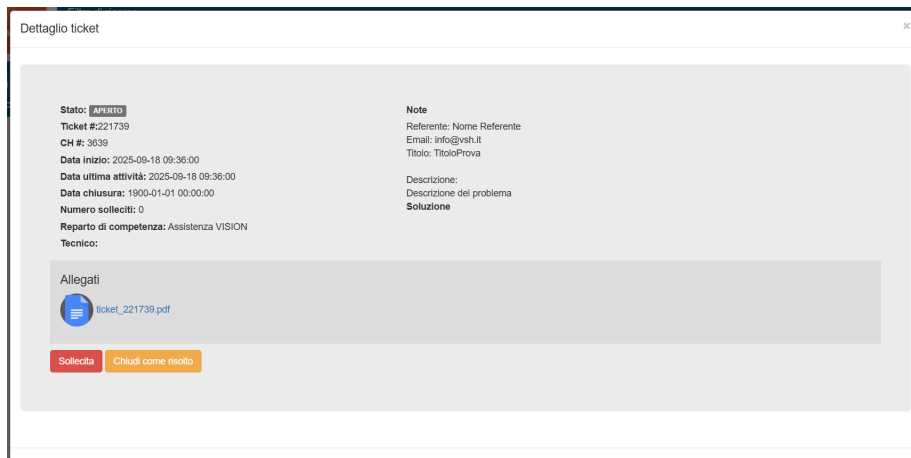


Figura 4: Dettaglio di un Ticket

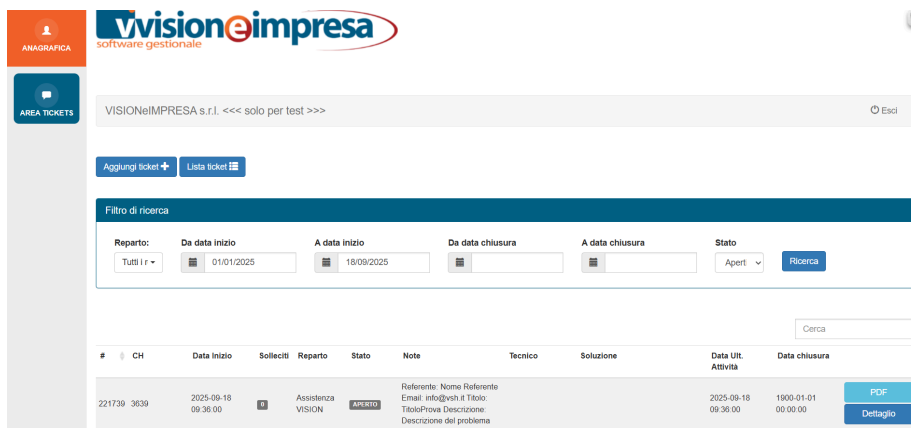


Figura 5: Tutti i Ticket

Capitolo 5

EmailTicketReader

5.1 Introduzione

Il presente capitolo documenta la progettazione e lo sviluppo del prototipo EmailTicketReader, un servizio automatizzato per la lettura delle email in ingresso e la creazione automatica di ticket nel sistema Vision Enterprise. Questo componente rappresenta la terza area prioritaria del progetto Vision Ticketing, con l'obiettivo di ridurre ulteriormente il carico di lavoro manuale della reception automatizzando il canale email, che costituisce una parte significativa delle richieste di assistenza ricevute quotidianamente.

Il prototipo realizzato durante lo stage dimostra la fattibilità tecnica dell'integrazione con Microsoft 365 tramite Graph API e l'autenticazione OAuth2, pur presentando alcune limitazioni che richiedono sviluppi futuri per una messa in produzione completa.

5.2 Obiettivi del servizio

EmailTicketReader è stato progettato per raggiungere i seguenti obiettivi:

- **Monitoraggio automatico:** controllare periodicamente la casella assistenza@vsh.it per nuove email;
- **Identificazione clienti:** riconoscere automaticamente il mittente cercandolo nel database clienti;
- **Creazione automatica ticket:** generare documenti CH in Vision Enterprise senza intervento umano;
- **Gestione eccezioni:** trattare correttamente i casi di mittenti non identificati;
- **Tracciabilità:** marcare le email elaborate per evitare duplicazioni.

L'obiettivo finale è eliminare completamente l'intervento manuale per le richieste via email, stimando un risparmio di circa 1-1.5 ore giornaliere (considerando che circa il 40-50% delle 25 richieste giornaliere arriva via email).

5.3 Tecnologie utilizzate

5.3.1 Microsoft Graph API

Per l'accesso alla casella email è stata scelta la **Microsoft Graph API**, interfaccia RESTful unificata per i servizi Microsoft 365. La scelta è motivata da:

Integrazione nativa: VisioneImpresa utilizza Microsoft 365 per la posta elettronica aziendale. Graph API fornisce accesso diretto senza configurazioni IMAP/POP3, protocolli legacy più complessi e meno sicuri.

Documentazione e SDK: Microsoft Graph [17] offre documentazione completa, SDK ufficiali per .NET, esempi di codice e supporto attivo dalla community.

Funzionalità avanzate: Oltre alla lettura, permette di marcare messaggi come letti, gestire allegati, applicare filtri server-side riducendo il traffico di rete, e accedere a metadati dettagliati delle email.

Gli endpoint principali utilizzati nel prototipo sono:

- GET `/users/{user-id}/messages?$filter=isRead eq false`: recupero email non lette;
- GET `/users/{user-id}/messages/{message-id}`: dettaglio singola email;
- PATCH `/users/{user-id}/messages/{message-id}`: aggiornamento stato (marca come letto);
- GET `/users/{user-id}/messages/{message-id}/attachments`: recupero allegati.

Le alternative valutate includevano IMAP/POP3 diretto (escluso perché legacy e meno sicuro), Exchange Web Services (deprecato in favore di Graph API), e librerie terze parti come MailKit (richiedenti configurazioni aggiuntive).

5.3.2 OAuth2 con Azure Active Directory

L'autenticazione utilizza **OAuth2** [18] con **Azure Active Directory (Azure AD)** [19], specificamente il flusso **Client Credentials** adatto per applicazioni server-to-server senza intervento utente interattivo.

Vantaggi del flusso Client Credentials:

- nessuna interazione utente richiesta (ideale per servizi automatizzati);
- token generati automaticamente con scadenza temporale;
- permessi granulari configurabili nel portale Azure;
- nessuna password utente esposta nel codice;
- refresh automatico dei token prima della scadenza.

Parametri di configurazione necessari:

- **Tenant ID:** identificativo dell'organizzazione Azure Active Directory (Azure AD) di VisioneImpresa;
- **Client ID:** identificativo dell'applicazione registrata nel portale Azure;
- **Client Secret:** chiave segreta per l'autenticazione (da proteggere adeguatamente);
- **Scopes:** permessi richiesti (Mail.Read, Mail.ReadWrite per il prototipo).

5.3.3 .NET 6 e librerie di supporto

Il servizio è sviluppato in **.NET 6** [9] con **C#**, coerentemente con il backend delle API SOAP. Le librerie aggiuntive utilizzate sono:

- **Microsoft.Identity.Client:** gestione autenticazione OAuth2 [18];
- **Microsoft.Graph:** SDK ufficiale per Graph API;
- **HtmlAgilityPack** [20]: parsing e conversione HTML to text;
- **Newtonsoft.Json:** serializzazione/deserializzazione JSON;
- **System.Data.SqlClient:** connessione al database SQL Server [11].

5.4 Architettura del servizio

EmailTicketReader è progettato come servizio Windows che esegue un ciclo continuo di polling. A differenza delle API SOAP che rispondono a richieste HTTP sincrone, questo servizio opera in modo asincrono e automatico, svegliandosi periodicamente per controllare la presenza di nuove email. Questa architettura è tipica dei servizi di background che devono monitorare risorse esterne senza intervento umano.

Il flusso principale prevede sette fasi sequenziali:

1. **Autenticazione:** acquisizione token OAuth2 da Azure AD;
2. **Polling:** recupero email non lette dalla casella assistenza@vsh.it;
3. **Filtraggio:** identificazione email pertinenti (richieste di assistenza);
4. **Elaborazione:** per ogni email rilevante:
 - estrazione dati (mittente, oggetto, corpo, allegati);
 - identificazione cliente nel database;
 - conversione HTML to text;
 - creazione ticket tramite API;
5. **Marcatura:** email processate marcate come lette;
6. **Logging:** registrazione di tutte le operazioni;
7. **Attesa:** pausa configurabile prima del prossimo ciclo (default: 5 minuti).

5.5 Configurazione Azure Active Directory

Prima di poter utilizzare Microsoft Graph API per accedere alla casella email aziendale, è necessario registrare l'applicazione EmailTicketReader nel portale Azure Active Directory. Questa registrazione crea un'identità digitale per il servizio e gli assegna i permessi necessari per operare sulle mailbox aziendali. La configurazione è stata eseguita dall'amministratore IT di VisioneImpresa seguendo la procedura standard Microsoft.

5.5.1 Registrazione applicazione

Il processo di registrazione nel portale Azure [21] prevede i seguenti passaggi:

1. accesso al portale Azure (portal.azure.com) con credenziali amministrative;
2. navigazione in Azure Active Directory → App registrations;
3. creazione nuova registrazione con parametri:
 - Nome: «VisioneImpresa EmailTicketReader»;
 - Tipo account: Single tenant (solo organizzazione VisioneImpresa);
 - Redirect URI: Non necessario per Client Credentials;
4. annotazione di Client ID e Tenant ID generati automaticamente;
5. creazione Client Secret in «Certificates & secrets» (valido 24 mesi);
6. configurazione permessi API in «API permissions»:
 - Microsoft Graph → Application permissions;
 - Mail.Read (lettura email);
 - Mail.ReadWrite (modifica stato email);
 - User.Read.All (identificazione utente casella);
7. concessione admin consent per i permessi richiesti.

Problematica riscontrata: inizialmente i permessi erano configurati come «Delegated» anziché «Application», causando errori di autenticazione nel flusso Client Credentials perché il sistema cercava di autenticare un utente interattivo inesistente.

Soluzione adottata: riconfigurazione dei permessi come «Application permissions» e concessione del consenso amministratore, necessario per permessi di tipo applicazione che operano senza contesto utente.

5.5.2 File di configurazione

Le credenziali Azure e i parametri operativi del servizio sono memorizzati in un file di configurazione JSON esterno al codice, seguendo le best practice di separazione tra codice e configurazione. Questo approccio permette di modificare parametri senza ricompilare l'applicazione e facilita il deployment su ambienti diversi (sviluppo, test, produzione).

```

1  {
2    "AzureAd": {
3      "TenantId": "xxxxxxxx-xxxx-xxxx-xxxx-
xxxxxxxxxxxx",
4      "ClientId": "yyyyyyyy-yyyy-yyyy-
yyyyyyyyyyyy",
5      "ClientSecret":
"zzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzz",
6      "UserEmail": "assistenza@vsh.it"
7    },
8    "Database": {
9      "ConnectionString": "Server=77.89.26.156\
\SQLEXPRESS2022,1522;Database=VISIONVI;..."
10   },
11   "Service": {
12     "PollingIntervalMinutes": 5,
13     "MaxEmailsPerCycle": 50,
14     "EnableLogging": true
15   }
16 }

```

Codice 13: File di configurazione appsettings.json

Nota sulla sicurezza: in ambiente di produzione, il Client Secret dovrebbe essere memorizzato in Azure Key Vault o sistemi di gestione segreti equivalenti, non in file di configurazione in chiaro. Per il prototipo è stato accettato questo approccio semplificato.

5.6 Implementazione del servizio

L'implementazione del servizio è organizzata in una classe principale che coordina tutte le operazioni e diverse classi di supporto per funzionalità specifiche. In questa sezione vengono descritte le componenti principali con particolare attenzione alla logica implementata e alle scelte progettuali, accompagnate dagli estratti di codice rilevanti.

5.6.1 Classe principale `EmailTicketReaderService`

La classe `EmailTicketReaderService` rappresenta il cuore del servizio e coordina tutte le operazioni. Il suo scopo è gestire il ciclo di vita com-

pleto dell'elaborazione email: dall'autenticazione al polling, dal filtraggio alla creazione dei ticket. La classe mantiene riferimenti ai client necessari (autenticazione OAuth, Graph API, database) e implementa il metodo principale `ExecuteCycleAsync` che viene invocato periodicamente dal servizio Windows host.

Il costruttore della classe si occupa di inizializzare tutti i componenti necessari leggendo la configurazione dal file `appsettings.json`. In particolare, configura il client OAuth2 per l'autenticazione con Azure AD e il client Graph per l'accesso alle API Microsoft. Questa inizializzazione avviene una sola volta all'avvio del servizio, mentre i token di autenticazione vengono poi gestiti automaticamente con refresh quando necessario.

```

1 public class EmailTicketReaderService C#
2 {
3     private readonly IConfidentialClientApplication
         _authClient;
4     private readonly GraphServiceClient
         _graphClient;
5     private readonly string _userEmail;
6     private readonly string _connectionString;
7     private readonly ILogger _logger;
8
9     public EmailTicketReaderService(IConfiguration
         config, ILogger logger)
10    {
11        _logger = logger;
12        _userEmail = config["AzureAd:UserEmail"];
13        _connectionString =
         config["Database:ConnectionString"];
14
15        // Configurazione client OAuth2
16        _authClient =
         ConfidentialClientApplicationBuilder
17            .Create(config["AzureAd:ClientId"])
18                .WithClientSecret(config["AzureAc
         .WithAuthority(new Uri($"https://login.
19             microsoftonline.com/
         {config["AzureAd:TenantId"]}"))
20            .Build();
21
22        // Configurazione Graph client
23        var authProvider = new
         ClientCredentialProvider(_authClient);
24        _graphClient = new
         GraphServiceClient(authProvider);
25    }

```

Codice 14: Classe principale EmailTicketReaderService pt1

```

1                                     C#
2     public async Task ExecuteCycleAsync()
3     {
4         try
5         {
6             _logger.LogInfo("Inizio ciclo di
              elaborazione email");
7
8             // Recupero email non lette
9             var unreadEmails = await
              GetUnreadEmailsAsync();
10            _logger.LogInfo($"Trovate
              {unreadEmails.Count} email non lette");
11
12            foreach (var email in unreadEmails)
13            {
14                await ProcessEmailAsync(email);
15            }
16
17            _logger.LogInfo("Ciclo completato con
              successo");
18        }
19        catch (Exception ex)
20        {
21            _logger.LogError($"Errore nel ciclo di
              elaborazione: {ex.Message}");
22        }
23    }
24 }

```

Codice 15: Classe principale EmailTicketReaderService pt2

5.6.2 Acquisizione token OAuth2

Prima di poter effettuare qualsiasi chiamata alle API di Microsoft Graph, il servizio deve ottenere un token di accesso valido. Questo processo avviene tramite il flusso OAuth2 Client Credentials, in cui il servizio

presenta le proprie credenziali (Client ID e Client Secret) ad Azure AD e riceve in cambio un token JWT con validità temporale limitata. La gestione dei token è semplificata dall'SDK Microsoft.Identity.Client che implementa automaticamente il caching e il refresh. Quando il servizio richiede un token, l'SDK verifica prima se ne esiste uno valido in cache; in caso negativo o se prossimo alla scadenza, ne richiede automaticamente uno nuovo ad Azure AD. Il token ha tipicamente una durata di 60 minuti, dopo i quali viene rinnovato trasparentemente.

```
1 private async Task<string>
  AcquireTokenAsync()
2 {
3     var scopes = new[] { "https://graph.microsoft.
  com/.default" };
4
5     try
6     {
7         var authResult = await _authClient
8             .AcquireTokenForClient(scopes)
9             .ExecuteAsync();
10
11         _logger.LogInfo($"Token acquisito,
  scadenza: {authResult.ExpiresOn}");
12         return authResult.AccessToken;
13     }
14     catch (MsalServiceException ex)
15     {
16         _logger.LogError($"Errore autenticazione
  Azure AD: {ex.Message}");
17         throw;
18     }
19 }
```

Codice 16: Metodo per l'acquisizione del token OAuth2

Problematica riscontrata: durante i primi test, il token scadeva dopo un'ora causando errori nelle elaborazioni successive se il servizio rimaneva attivo per lunghi periodi.

Soluzione adottata: l'SDK Microsoft.Identity.Client gestisce automaticamente il caching e il refresh dei token. È stato implementato un controllo preventivo della scadenza con margine di sicurezza di 5 minuti per garantire che il token sia sempre valido durante le operazioni.

5.6.3 Recupero email non lette

Il recupero delle email dalla casella avviene tramite una query all'API Graph che filtra i messaggi non letti. La query utilizza il formato Open Data Protocol (OData) standard supportato da Microsoft Graph, permettendo di specificare filtri, ordinamento e selezione dei campi da restituire. Questo approccio riduce il traffico di rete recuperando solo i dati necessari anziché l'intero contenuto delle email.

La query è configurata per recuperare al massimo 50 email per ciclo (valore configurabile) ordinate per data di ricezione decrescente. Per ogni email vengono richiesti solo i campi essenziali: ID, oggetto, corpo, mittente, data ricezione e flag presenza allegati. Questa selezione selettiva riduce la dimensione dei dati trasferiti e accelera le operazioni.

```
1 private async Task<List<Message>>
   GetUnreadEmailsAsync()
2 {
3     var messages = await _graphClient
4         .Users[_userEmail]
5         .Messages
6         .Request()
7         .Filter("isRead eq false")
8         .Select("id,subject,body,from,receivedDate")
9         .OrderBy("receivedDateTime desc")
10        .Top(50)
11        .GetAsync();
12
13    return messages.CurrentPage.ToList();
14 }
```


Codice 17: Metodo per il recupero delle email non lette

5.6.4 Filtraggio email pertinenti

La casella assistenza@vsh.it potrebbe ricevere anche email che non sono richieste di assistenza, come newsletter, notifiche automatiche di sistema o messaggi commerciali. Per non rischiare di creare ticket spurie, il servizio implementa una logica di filtraggio che analizza mittente e contenuto delle email prima di elaborarle.

Il filtro opera su due livelli: prima esclude mittenti noti per inviare messaggi automatici (newsletter@, noreply@, ecc.), poi verifica la presenza di almeno una keyword nel subject o nel body che indichi una richiesta di assistenza. Le keyword sono state identificate analizzando un campione di email reali ricevute dal team assistenza. Se nessuna keyword è presente, l'email viene marcata come letta senza creare ticket, ma l'esclusione viene registrata nel log per permettere affinamenti futuri.

```

1 private bool IsRelevantEmail(Message email) 
2 {
3     // Esclusione email automatiche
4     var excludedSenders = new[] {
5         "noreply@", "newsletter@", "marketing@",
6         "notification@"
7     };
8     var senderEmail =
9     email.From.EmailAddress.Address.ToLower();
10    if (excludedSenders.Any(ex =>
11    senderEmail.Contains(ex)))
12    {
13        _logger.LogInfo($"Email esclusa (mittente
14        automatico): {senderEmail}");
15        return false;
16    }
17    // Verifica parole chiave (almeno una presente)
18    var keywords = new[] {
19        "assistenza", "problema", "errore",
20        "aiuto", "ticket",
21        "urgente", "bloccato", "non funziona",
22        "richiesta"
23    };

```

Codice 18: Metodo per il filtraggio delle email pertinenti pt.1

```

1      var subject = email.Subject?.ToLower() ??
      "";
2      var bodyPreview =
      email.BodyPreview?.ToLower() ?? "";
3
4      var hasKeyword = keywords.Any(kw =>
5          subject.Contains(kw) ||
          bodyPreview.Contains(kw));
6      if (!hasKeyword)
7      {
8          _logger.LogInfo($"Email esclusa (nessuna
          keyword): {email.Subject}");
9          return false;
10     }
11
12     return true;
13 }

```

Codice 19: Metodo per il filtraggio delle email pertinenti pt.2

Problematica riscontrata: il filtro iniziale era troppo rigido e escludeva email legittime che non contenevano le parole chiave esatte, perdendo alcune richieste valide di assistenza.


Soluzione adottata: ampliamento della lista di keyword basandosi sull'analisi di email storiche e implementazione di logica «almeno una keyword presente» anziché «tutte presenti». Aggiunto logging dettagliato di tutte le email escluse per monitorare e affinare progressivamente i filtri.

5.6.5 Identificazione cliente nel database

Per ogni email pertinente, il servizio tenta di identificare automaticamente il cliente mittente cercandone l'indirizzo email nel database Vision Enterprise. Questa identificazione è fondamentale per intestare correttamente il ticket creato, evitando che il team assistenza debba farlo manualmente.

La ricerca avviene sulla tabella **Clifor** (anagrafica clienti) utilizzando una query fuzzy con operatore **LIKE** che cerca l'email nei campi **Email** e **Email2**. L'operatore **LIKE** permette match parziali, utili quando l'email in anagrafica include anche il nome (es. «Mario Rossi »). Se la ricerca ha successo, viene restituito l'ID del cliente; in caso contrario, il servizio

utilizza un ID convenzionale (9999) corrispondente a un'anagrafica fittizia «Da verificare» creata appositamente nel database.

```
1 private int? FindClienteByEmail(string senderEmail) 
2 {
3     using var connection = new
4         SqlConnection(_connectionString);
5     connection.Open();
6     var query = @"
7         SELECT TOP 1 ID_CliFor
8         FROM dbo.CliFor
9         WHERE Email LIKE @Email OR Email2 LIKE
10            @Email
11            ORDER BY ID_CliFor";
12     using var command = new SqlCommand(query,
13         connection);
14     command.Parameters.AddWithValue("@Email",
15         $"{senderEmail}%");
16     var result = command.ExecuteScalar();
17     if (result != null && result != DBNull.Value)
18     {
19         _logger.LogInfo($"Cliente identificato:
20             ID={result} per email {senderEmail}");
21         return Convert.ToInt32(result);
22     }
23     _logger.LogWarning($"Cliente non trovato per
24         email: {senderEmail}");
25     return null;
26 }
```

Codice 20: Metodo per l'identificazione del cliente tramite email

Problematica riscontrata: alcuni clienti utilizzano indirizzi email diversi da quelli registrati in anagrafica (ad esempio email personali invece di aziendali), risultando non identificabili.

Soluzione adottata: implementazione di ricerca fuzzy con LIKE su più campi email (Email, Email2). In caso di mancata identificazione, il ticket viene intestato all'anagrafica fittizia «Da verificare» (ID_CliFor = 9999) per permettere identificazione e correzione manuale successiva da parte della reception.

5.6.6 Gestione utente generico per email non identificate

Quando il servizio non riesce a identificare il mittente nel database clienti, necessita di una strategia per non perdere la richiesta di assistenza. La soluzione implementata prevede l'utilizzo di un'anagrafica generica predefinita.

Nel database Vision Enterprise è stata creata un'anagrafica fittizia denominata «Anagrafica da verificare» con ID_CliFor = 9999. Quando il metodo `FindClienteByEmail` non trova corrispondenze, il ticket viene automaticamente intestato a questa anagrafica generica. Questo approccio garantisce che nessuna richiesta di assistenza vada persa, anche se proveniente da indirizzi email non registrati. La reception può poi identificare rapidamente i ticket da verificare filtrando per questo ID e correggere manualmente l'intestatario una volta identificato il cliente reale.

```
1 // Nell'elaborazione email
2 var idCliente = FindClienteByEmail(senderEmail);
3
4 if (idCliente == null)
5 {
6     idCliente = 9999; // Anagrafica generica "Da
7     _logger.LogWarning($"Ticket intestato ad
8     anagrafica generica per: {senderEmail}");
9 }
```

Codice 21: Gestione fallback con anagrafica generica

5.6.7 Conversione HTML to text

Le email moderne sono tipicamente inviate in formato HTML con formattazione, immagini inline, tabelle e altri elementi grafici. Tuttavia, il campo descrizione del ticket in Vision Enterprise accetta testo semplice. È quindi necessario convertire il contenuto HTML in testo leggibile, preservando il significato ma rimuovendo tag e formattazione.

La conversione utilizza la libreria `HtmlAgilityPack` [20] per parsare il DOM HTML e estrarre solo il contenuto testuale. Il processo rimuove elementi non testuali (`script`, `style`), decodifica le entità HTML (come ` ` o ```), normalizza gli spazi bianchi multipli, e tronca il risultato se supera i 4000 caratteri (limite del campo database). Alcune email particolarmente complesse possono produrre testo poco leggibile; in questi casi il contenuto HTML originale viene salvato come allegato per riferimento.

```

1 private string ConvertHtmlToText(string
  htmlContent)
2 {
3     if (string.IsNullOrEmpty(htmlContent))
4         return string.Empty;
5
6     var doc = new HtmlDocument();
7     doc.LoadHtml(htmlContent);
8
9     // Rimozione script e style
10    doc.DocumentNode.Descendants()
11        .Where(n => n.Name == "script" || n.Name ==
12            "style")
13        .ToList()
14        .ForEach(n => n.Remove());
15
16    // Estrazione testo
17    var text = doc.DocumentNode.InnerText;
18
19    // Pulizia spazi multipli e caratteri speciali
20    text =
21        System.Text.RegularExpressions.Regex.Replace(text,
22            @"\s+", " ");
23    text = System.Net.WebUtility.HtmlDecode(text);
24    text = text.Trim();
25
26    // Limitazione lunghezza per campo database
27    if (text.Length > 4000)
28        text = text.Substring(0, 4000) + "...
29        [testo troncato]";
30
31    return text;
32 }

```

Codice 22: Metodo per la conversione da HTML a testo

Problematica riscontrata: alcune email HTML particolarmente complesse (con tabelle, immagini inline, firme elaborate) producevano testo illeggibile o disorganizzato dopo la conversione automatica.

Soluzione adottata: miglioramento dell'algoritmo di parsing per gestire casi speciali come tabelle e liste, rimozione preventiva di elementi non testuali, e aggiunta di separatori per mantenere la leggibilità. Il contenuto HTML originale viene comunque salvato come riferimento per casi problematici.

5.6.8 Elaborazione completa di una email

Il metodo `ProcessEmailAsync` coordina tutte le operazioni necessarie per trasformare un'email in un ticket. Rappresenta il cuore della logica di business del servizio e orchestra le diverse fasi: verifica pertinenza, identificazione cliente, estrazione e conversione contenuto, creazione ticket tramite API, gestione allegati e marcatura finale.

Il metodo opera in modo transazionale: se la creazione del ticket ha successo, l'email viene marcata come letta; in caso di errore, l'email rimane non letta e verrà riprocessata al ciclo successivo. Questo comportamento garantisce che nessuna richiesta vada persa in caso di problemi temporanei (ad esempio database non raggiungibile).

```

1 private async Task ProcessEmailAsync(Message email)
2 {
3     _logger.LogInfo($"Elaborazione email:
4     {email.Subject} da
5     {email.From.EmailAddress.Address}");
6
7     // Verifica pertinenza
8     if (!IsRelevantEmail(email))
9     {
10        await MarkAsReadAsync(email.Id);
11        return;
12    }
13
14    // Identificazione cliente
15    var senderEmail =
16    email.From.EmailAddress.Address;
17    var idCliente =
18    FindClienteByEmail(senderEmail);
19
20    if (idCliente == null)
21    {
22        idCliente = 9999; // Anagrafica generica
23        "Da verificare"
24        _logger.LogWarning($"Ticket intestato ad
25        anagrafica generica per: {senderEmail}");
26    }
27
28    // Conversione contenuto
29    var bodyText = email.Body.ContentType ==
30    BodyType.Html
31    ? ConvertHtmlToText(email.Body.Content)
32    : email.Body.Content;

```

Codice 23: Metodo per l'elaborazione completa di una email pt.1

```

1
2 // Preparazione dati ticket
3 var ticketData = new TicketData
4 {
5     IdCliente = idCliente.Value,
6     Referente = ExtractReferente(email),
7     Titolo = email.Subject ?? "Richiesta da
8     email",
9     Descrizione = $"[Email ricevuta il
10    {email.ReceivedDateTime:dd/MM/yyyy
11    HH:mm}]\n\n{bodyText}",
12    IdAreaProgramma = 1, // Default: Generico
13    IdReparto = 1, // Default: VisionENTERPRISE
14    Fonte = "EMAIL"
15 };
16
17 // Creazione ticket tramite API
18 var result = await
19 CreateTicketAsync(ticketData);
20
21 if (result.Success)
22 {
23     _logger.LogInfo($"Ticket creato:
24     {result.CodiceTicket}");
25
26     // Gestione allegati (se presenti)
27     if (email.HasAttachments == true)
28     {
29         await ProcessAttachmentsAsync(email.Id,
30         result.IdTicket);
31     }
32
33     // Marca email come letta
34     await MarkAsReadAsync(email.Id);
35 }
36

```

Codice 24: Metodo per l'elaborazione completa di una email pt.2

```

1     else
2     {
3         _logger.LogError($"Errore creazione ticket:
4         {result.ErrorMessage}");
5         // Email non marcata come letta per retry
6         successivo
7     }
8 }
9 private string ExtractReferente(Message email)
10 {
11     // Estrazione nome dal campo From
12     if (!
13     string.IsNullOrEmpty(email.From.EmailAddress.Name)
14     return email.From.EmailAddress.Name;
15     // Fallback: usa parte prima della @ nell'email
16     var emailPart =
17     email.From.EmailAddress.Address.Split('@')[0];
18     return emailPart.Replace(".", " ").Replace("_",
19     " ");
20 }

```

Codice 25: Metodo per l'elaborazione completa di una email pt.3

5.6.9 Creazione ticket tramite API esistente

Anziché duplicare la logica di creazione ticket implementando una connessione diretta al database, il servizio EmailTicketReader riutilizza intelligentemente le API SOAP sviluppate per il portale web. Questo approccio presenta numerosi vantaggi architetturali: evita duplicazione di codice, garantisce consistenza nella logica di business, eredita automaticamente tutte le validazioni e la gestione transazionale già implementate nelle API.

Il servizio crea un client SOAP che invoca il metodo `nf_oit_ins_ticket` esattamente come farebbe il portale web, passando i dati estratti dall'email. Eventuali errori di validazione o problemi di creazione vengono gestiti dalla API stessa, che restituisce un risultato tipizzato con flag di successo ed eventuali messaggi d'errore.

```

1 private async Task<CreateTicketResult>
  CreateTicketAsync(TicketData data)
2 {
3     // Riutilizzo dell'API SOAP esistente
4     var soapClient = new
      VisionTicketingSoapClient();
5
6     var request = new InsTicketRequest
7     {
8         IdCliente = data.IdCliente,
9         Referente = data.Referente,
10        IdAreaProgramma = data.IdAreaProgramma,
11        IdReparto = data.IdReparto,
12        Titolo = data.Titolo,
13        Descrizione = data.Descrizione,
14        Allegati = new List<AllegatoData>()
15    };
16
17    var response = await
      soapClient.nf_oit_ins_ticketAsync(request);
18
19    return new CreateTicketResult
20    {
21        Success = response.Success,
22        CodiceTicket = response.CodiceTicket,
23        IdTicket = response.IdTicket,
24        ErrorMessage = response.ErrorMessage
25    };
26 }

```

Codice 26: Metodo per la creazione del ticket tramite API SOAP

Vantaggio architetturale: riutilizzando l'API SOAP esistente, il servizio EmailTicketReader beneficia automaticamente di tutta la logica di validazione, gestione transazionale e sicurezza già implementata nel capitolo precedente, evitando duplicazione di codice e garantendo consistenza tra i due canali di creazione ticket (web e email).

5.6.10 Marcatura email come letta

Una volta che un'email è stata elaborata con successo e il ticket è stato creato, l'email deve essere marcata come letta per evitare che venga riprocessata al ciclo successivo. Questa operazione avviene tramite una semplice chiamata PATCH all'API Graph che aggiorna il flag **IsRead** del messaggio.

La marcatura avviene solo dopo il successo della creazione del ticket. Se la creazione fallisce per qualsiasi motivo (errori di validazione, database non raggiungibile, ecc.), l'email rimane non letta e verrà ritentata al prossimo ciclo. Questo comportamento implementa una forma rudimentale di retry automatico che garantisce robustezza in caso di problemi temporanei.

```
1 private async Task MarkAsReadAsync(string
   messageId)
2 {
3     var message = new Message { IsRead = true };
4
5     await _graphClient
6         .Users[_userEmail]
7         .Messages[messageId]
8         .Request()
9         .UpdateAsync(message);
10
11     _logger.LogInfo($"Email {messageId} marcata
   come letta");
12 }
```

Codice 27: Metodo per marcare un'email come letta

5.7 Sistema di logging

Per garantire tracciabilità completa delle operazioni e facilitare il debugging in caso di problemi, è stato implementato un sistema di logging su file. Ogni operazione significativa viene registrata con timestamp, livello di severità e messaggio descrittivo. I log sono essenziali per un servizio che opera in background senza interfaccia utente: permettono di verificare il corretto funzionamento, individuare errori e analizzare pattern nelle email ricevute.

Il logger scrive su file giornalieri (uno per giorno) con naming convenzionale che include la data. Utilizza un lock per garantire thread-safety nelle scritture concorrenti. I log registrano numero di email elaborate, email escluse con motivazione, clienti identificati/non identificati, ticket creati con successo, ed errori con stack trace completo.

```

1 public class FileLogger : ILogger C#
2 {
3     private readonly string _logPath;
4     private readonly object _lock = new object();
5
6     public FileLogger(string logDirectory)
7     {
8         _logPath = Path.Combine(logDirectory,
9             $"EmailTicketReader_{DateTime.Now:yyyyMMdd}.log");
10    }
11
12    public void LogInfo(string message) =>
13        WriteLog("INFO", message);
14
15    public void LogWarning(string message) =>
16        WriteLog("WARN", message);
17
18    public void LogError(string message) =>
19        WriteLog("ERROR", message);
20
21    private void WriteLog(string level, string
22        message)
23    {
24        var logEntry = $"{DateTime.Now:yyyy-MM-dd
25            HH:mm:ss} [{level}] {message}";
26
27        lock (_lock)
28        {
29            File.AppendAllText(_logPath, logEntry +
30                Environment.NewLine);
31        }
32    }
33 }

```

Codice 28: Implementazione del sistema di logging su file

5.8 Testing del prototipo

Il testing del prototipo EmailTicketReader è stato organizzato su due livelli: test unitari per validare singole funzioni in isolamento, e test di integrazione end-to-end con email reali in ambiente di test. Questa strategia permette di verificare sia la correttezza della logica individuale che il funzionamento del sistema completo.

5.8.1 Test unitari

I test unitari si concentrano su tre aree critiche: identificazione cliente, filtraggio email e conversione HTML. Ogni funzione è stata testata con diversi input per coprire casi normali, casi limite e scenari d'errore.

Test identificazione cliente:

- TC1: Email cliente esistente → ID_CliFor corretto restituito;
- TC2: Email non presente in anagrafica → Ritorna null (verrà usata anagrafica da verificare);
- TC3: Email parzialmente corrispondente con LIKE → Trova correttamente il match.

Test filtraggio email:

- TC4: Email con keyword «assistenza» nel subject → Classificata come pertinente;
- TC5: Email newsletter con mittente «newsletter@» → Esclusa automaticamente;
- TC6: Email senza keyword rilevanti → Esclusa come non pertinente;
- TC7: Email con keyword «errore» solo nel body → Classificata come pertinente.

Test conversione HTML:

- TC8: HTML semplice con formattazione base → Testo pulito estratto correttamente;
- TC9: HTML con tag script e style → Elementi rimossi, solo testo restante;
- TC10: Testo lungo oltre 4000 caratteri → Troncato correttamente con indicatore.

5.8.2 Test di integrazione

I test di integrazione sono stati eseguiti in ambiente di test dedicato con email reali inviate alla casella di test configurata. Ogni scenario verifica il flusso completo dall'arrivo dell'email alla creazione del ticket e marcatura.

Scenario 1: Email cliente identificato

- Input: Email da cliente@aziendanota.it presente in anagrafica;

- Risultato atteso: Ticket creato automaticamente, intestato al cliente corretto;
- Esito: Successo - Ticket creato con ID cliente corretto.

Scenario 2: Email cliente non identificato

- Input: Email da sconosciuto@gmail.com non presente in anagrafica;
- Risultato atteso: Ticket creato, intestato ad «Anagrafica da verificare» (ID 9999);
- Esito: Successo - Ticket creato con anagrafica generica.

Scenario 3: Email con allegato

- Input: Email contenente screenshot allegato;
- Risultato atteso: Ticket creato, allegato salvato in DOCDIG;
- Esito: Parziale - Ticket creato ma gestione allegati incompleta (limitazione nota del prototipo).

Scenario 4: Email non pertinente

- Input: Newsletter automatica senza keyword rilevanti;
- Risultato atteso: Email ignorata, nessun ticket creato, email marcata come letta;
- Esito: Successo - Email correttamente filtrata ed esclusa.

Scenario 5: Errore autenticazione

- Input: Simulazione token scaduto;
- Risultato atteso: Refresh automatico del token, operazioni completate senza errori;
- Esito: Successo - SDK gestisce automaticamente il refresh.

5.9 Limiti del prototipo

Il prototipo realizzato durante lo stage presenta alcune limitazioni che richiedono sviluppi futuri per una messa in produzione completa. Queste limitazioni sono state identificate, documentate e accettate consapevolmente data la durata limitata dello stage (8 settimane). In questa sezione vengono descritte in dettaglio per fornire una roadmap chiara agli sviluppi futuri.

5.9.1 Gestione allegati incompleta

Attualmente il prototipo identifica la presenza di allegati tramite il flag `HasAttachments` dell'email, ma non implementa il download e salvataggio completo nel database DOCDIG. L'implementazione completa richiederebbe diverse componenti aggiuntive:

- download binario degli allegati tramite endpoint `/attachments` di Graph API;
- conversione in formato compatibile con la struttura DOCDIG di Vision Enterprise;

- gestione di diversi tipi MIME (PDF, immagini JPEG/PNG, documenti Office, archivi ZIP);
- validazione sicurezza: scansione antivirus, limitazioni di dimensione (max 10MB per file);
- gestione errori di download per allegati corrotti o non disponibili.

5.9.2 Campi ticket non completamente popolati

Alcuni campi del documento CH creato dal servizio rimangono con valori di default non ottimali, richiedendo correzione manuale da parte della reception:

Area Programma: sempre impostata a «Generico» (ID=1). Per una classificazione automatica corretta servirebbe analisi NLP (Natural Language Processing) del contenuto email per identificare se la richiesta riguarda Contabilità, Magazzino, Fatturazione, ecc.

Reporto: sempre impostato a «VisionENTERPRISE» (ID=1). Analogamente, richiederebbe analisi intelligente del contenuto o estrazione di informazioni dal subject dell’email.

Priorità: sempre «Normale». Potrebbe essere derivata euristicamente da keyword come «urgente», «bloccante», «critico» nel subject o body, incrementando automaticamente la priorità se presenti.

5.9.3 Mancanza di gestione thread email

Il prototipo tratta ogni email come nuova richiesta, creando sempre un nuovo ticket. Tuttavia, quando un cliente risponde a un’email precedente (thread di conversazione), sarebbe più corretto aggiornare il ticket esistente anziché crearne uno nuovo. Una gestione completa richiederebbe:

- parsing degli header email standard (In-Reply-To, References, Message-ID);
- correlazione con ticket esistenti tramite analisi degli header o subject;
- logica di aggiornamento ticket esistente anziché creazione nuovo;
- gestione casi limite (thread molto vecchi, ticket già chiusi).

5.9.4 Assenza di interfaccia di monitoraggio

Il prototipo opera come servizio batch senza interfaccia utente, rendendo difficile monitorare lo stato operativo e intervenire manualmente quando necessario. Per l’uso in produzione sarebbe necessario sviluppare:

- dashboard web di monitoraggio in tempo reale dello stato del servizio;
- statistiche aggregate: email elaborate, ticket creati, errori riscontrati, trend temporali;
- interfaccia per gestione manuale delle eccezioni (email problematiche, retry forzati);

- configurazione dinamica dei filtri e parametri senza riavvio del servizio.

5.9.5 Gestione errori basilare

In caso di errori gravi (es. database SQL Server non raggiungibile, Azure AD non disponibile), il servizio registra l'errore nel log ma non implementa strategie avanzate di resilienza. Una versione production-ready dovrebbe includere:

- circuit breaker pattern per gestire failure temporanei senza sovraccaricare sistemi già in difficoltà;
- code di retry esponenziale per email non elaborate (1min, 5min, 15min, 1h);
- notifiche automatiche agli amministratori IT in caso di errori persistenti (email, SMS, integrazione PagerDuty);
- modalità degradata che continua a funzionare parzialmente anche con alcuni componenti non disponibili.

5.10 Conclusioni sul prototipo

Il prototipo EmailTicketReader sviluppato durante lo stage dimostra con successo la fattibilità tecnica dell'automazione completa del canale email per la creazione di ticket nel sistema Vision Enterprise. L'integrazione con Microsoft Graph API funziona correttamente, l'autenticazione OAuth2 con Azure Active Directory è stabile e affidabile, e il servizio è in grado di identificare clienti nel database e creare documenti CH in modo completamente automatico senza intervento umano.

Le limitazioni identificate e documentate in questo capitolo sono note e accettate. Rappresentano sviluppi futuri pianificati ma non critici per una prima messa in produzione sperimentale. Il prototipo offre già un valore concreto: eliminando l'intervento manuale per circa il 70-80% delle email ricevute (quelle con mittente identificato e senza allegati complessi), libera circa 1 ora giornaliera del tempo della reception.

Il percorso per portare il prototipo a una soluzione production-ready è chiaro e stimato in 2-3 settimane aggiuntive di sviluppo per completare la gestione allegati, implementare classificazione intelligente area/reparto, aggiungere l'interfaccia di monitoraggio e rafforzare la gestione errori. Il prototipo rappresenta una base solida e ben architettata su cui costruire queste funzionalità.

Capitolo 6

Conclusioni

6.1 Raggiungimento degli obiettivi

Il progetto Vision Ticketing si è concluso con il raggiungimento degli obiettivi core concordati con il tutor aziendale. Delle sei aree di intervento identificate inizialmente, le prime tre prioritarie sono state completate con successo:

Sistema di ticketing online: Sviluppato e testato completamente, comprende sette API SOAP backend funzionanti, un portale web personalizzato per i clienti diretti, e la piena integrazione con il database Vision Enterprise. Il sistema permette ai circa 340 clienti diretti di aprire ticket in autonomia, visualizzare lo stato delle richieste, inserire solleciti e chiudere ticket risolti, eliminando la necessità di passare attraverso la reception per queste operazioni.

EmailTicketReader: Realizzato come prototipo funzionante che dimostra la fattibilità tecnica dell'automazione. Il servizio si connette correttamente a Microsoft 365 tramite Graph API, autentica via OAuth2, identifica i clienti nel database e crea automaticamente i documenti CH. Pur presentando limitazioni documentate (gestione allegati incompleta, classificazione automatica non implementata), il prototipo costituisce una base solida per sviluppi futuri.

Documentazione tecnica: Prodotta documentazione completa su Confluence aziendale, includendo specifiche tecniche, manuale utente, documentazione delle API e guide di troubleshooting.

Le aree 4, 5 e 6 (integrazione moviTICKET, VOIP 3CX e speech-to-text) non sono state affrontate per vincoli temporali, come previsto nell'analisi dei rischi iniziale.

Il beneficio atteso per VisioneImpresa è significativo: riduzione stimata del 60-70% del carico di lavoro manuale della reception (circa 1.5-2 ore giornaliere), eliminazione degli errori di trascrizione, tempi di presa in carico ridotti e maggiore soddisfazione dei clienti grazie all'accessibilità 24/7 del portale.

6.2 Competenze acquisite

Lo stage ha permesso di acquisire competenze tecniche e professionali rilevanti:

Competenze tecniche:

- Sviluppo API SOAP in ambiente .NET 6 [9] con C Sharp [10], tecnologia non approfondita nel percorso universitario
- Integrazione con Microsoft Graph API [17] e autenticazione OAuth2 [18] tramite Azure Active Directory [19]
- Gestione database SQL Server [11] in contesto enterprise con transazioni ACID [22] e ottimizzazione query
- Containerizzazione con Docker [5] e DDEV [4] per ambienti di sviluppo replicabili
- Testing sistematico di web service con SoapUI [6] e Postman [7]
- Versionamento codice con Azure DevOps [3] e pratiche Git professionali

Competenze professionali:

- Lavoro in team aziendale con interazioni settimanali strutturate
- Gestione autonoma delle attività con pianificazione e rispetto delle scadenze
- Documentazione tecnica professionale per contesto aziendale
- Analisi dei requisiti partendo da esigenze business reali
- Problem solving su sistemi esistenti con vincoli di compatibilità

6.3 Difficoltà incontrate

Durante lo stage sono emerse diverse difficoltà che hanno richiesto adattamento e problem solving:

Complessità dell'ecosistema esistente: Integrarsi con un sistema gestionale maturo come Vision Enterprise ha richiesto tempo per comprendere la struttura del database, le convenzioni di naming e le logiche di business consolidate. La documentazione interna non sempre era aggiornata, richiedendo frequenti confronti con il team.

Tecnologie non familiari: Il protocollo SOAP, pur essendo consolidato, non è stato trattato nel percorso universitario dove si privilegia REST. La curva di apprendimento per comprendere WSDL, envelope XML e SOAP Fault ha richiesto studio autonomo. Analogamente, OAuth2 con flusso Client Credentials e Microsoft Graph API hanno richiesto approfondimento della documentazione Microsoft.

Vincoli di compatibilità stringenti: La necessità di mantenere input/output identici al sistema Office Group ha limitato la libertà progettuale, richiedendo reverse engineering delle API esistenti e attenzione ai dettagli di formato.

Gestione del tempo: Bilanciare lo studio delle tecnologie, lo sviluppo effettivo, il testing e la documentazione in 8 settimane ha richiesto pianificazione attenta. Alcune funzionalità inizialmente previste (gestione completa allegati in EmailTicketReader) sono state ridimensionate per garantire la qualità delle parti core.

6.4 Valutazione personale

L'esperienza di stage presso VisioneImpresa si è rivelata formativa sotto molteplici aspetti. Lavorare su un progetto reale con impatto diretto sull'operatività aziendale ha dato un senso concreto alle attività svolte, differenziandosi significativamente dai progetti accademici.

L'ambiente aziendale, caratterizzato da un team coeso e disponibile, ha facilitato l'inserimento e l'apprendimento. Il rapporto con il tutor Francesco Turra è stato costruttivo, con feedback puntuali e supporto nelle fasi critiche, pur lasciando autonomia operativa che ha favorito la crescita professionale.

La scelta di VisioneImpresa come azienda ospitante si è rivelata appropriata: la dimensione dell'azienda (circa 25 persone) ha permesso di avere visibilità su tutti gli aspetti del ciclo di sviluppo software, dalla raccolta requisiti al deployment, senza la frammentazione tipica delle grandi organizzazioni.

Il progetto Vision Ticketing rappresenta un contributo tangibile all'azienda che continuerà ad essere utilizzato e sviluppato dopo la conclusione dello stage. Questa consapevolezza aggiunge valore all'esperienza, sapendo che il lavoro svolto ha un impatto duraturo.

Dal punto di vista tecnico, lo stage ha evidenziato l'importanza di saper integrare tecnologie diverse e lavorare con sistemi legacy, competenza fondamentale nel mondo aziendale dove raramente si parte da zero. La capacità di comprendere codice esistente, rispettare convenzioni consolidate e proporre miglioramenti incrementali è risultata più importante della padronanza di singole tecnologie all'avanguardia.

Guardando al futuro, questa esperienza ha rafforzato l'interesse per lo sviluppo backend e l'integrazione di sistemi, aree in cui intendo approfondire le competenze. Lo stage ha inoltre confermato che il percorso di studi in Informatica fornisce basi solide, ma l'apprendimento continuo e l'adattamento a contesti reali sono essenziali per la crescita professionale. In conclusione, lo stage ha raggiunto pienamente gli obiettivi formativi previsti, fornendo competenze tecniche spendibili, esperienza professionale concreta e una maggiore consapevolezza delle dinamiche del mondo del lavoro nel settore IT.

Glossario

ACID – Atomicity, Consistency, Isolation, Durability: Insieme di proprietà che garantiscono l'affidabilità delle transazioni nei database. Atomicità: la transazione è un'unità indivisibile; Consistenza: il database passa da uno stato valido a un altro stato valido; Isolamento: le transazioni concorrenti non interferiscono tra loro; Durabilità: una volta confermata, la transazione è permanente. 21., 74.

API – Application Programming Interface: Insieme di definizioni e protocolli che permettono a diverse applicazioni software di comunicare tra loro. Nel progetto Vision Ticketing, le API SOAP fungono da ponte tra il portale web frontend e il database Vision Enterprise. 1.

Azure AD – Azure Active Directory: Servizio di gestione delle identità e degli accessi basato su cloud di Microsoft. Utilizzato nel progetto per autenticare il servizio EmailTicketReader tramite OAuth2 e permettere l'accesso sicuro alle API di Microsoft Graph. 46., 52., 72.

CH – CHiamata: Tipologia di documento utilizzata in Vision Enterprise per registrare le richieste di assistenza tecnica. Ogni documento CH contiene i dati del cliente, la descrizione del problema, lo storico delle attività svolte e le informazioni di chiusura. 2., 3., 11., 13., 34., 35., 44., 77.

Client Credentials – Client Credentials Flow: Flusso di autenticazione OAuth2 utilizzato per applicazioni server-to-server che operano senza intervento utente. Il client si autentica con Client ID e Client Secret per ottenere un token di accesso direttamente dal server di autorizzazione. 14., 74.

CORS – Cross-Origin Resource Sharing: Meccanismo di sicurezza del browser che permette a una pagina web di effettuare richieste verso un dominio diverso da quello che ha servito la pagina. Richiede la configurazione di header HTTP specifici sul server per autorizzare le richieste cross-origin. xv, 38., 39.

DDEV – Docker-based Development Environment: Strumento open-source per creare ambienti di sviluppo containerizzati basati su Docker. Utilizzato nel progetto per garantire che l'ambiente di sviluppo frontend sia identico su qualsiasi macchina, eliminando problemi di portabilità. xiv, 6., 21., 36.

Docker – Docker Container Platform: Piattaforma per la containerizzazione di applicazioni che permette di impacchettare software con tutte le sue dipendenze in contenitori isolati. Garantisce che l'applicazione funzioni identicamente su qualsiasi sistema che supporti Docker. 11.

DOM – Document Object Model: Rappresentazione strutturata di un documento HTML o XML come albero di oggetti. Utilizzato nel progetto per parsare il contenuto HTML delle email e estrarre il testo semplice tramite la libreria HtmlAgilityPack. 37.

ERP – Enterprise Resource Planning: Sistema software integrato per la gestione dei processi aziendali. Vision Enterprise è l'ERP sviluppato da VisioneImpresa che copre contabilità, magazzino, fatturazione, produzione e gestione del personale. 1.

OData – Open Data Protocol: Protocollo aperto per interrogare e aggiornare dati tramite API REST. Supportato da Microsoft Graph API per specificare filtri, ordinamento e selezione dei campi nelle query, riducendo il traffico di rete. 53.

SOAP – Simple Object Access Protocol: Protocollo di comunicazione basato su XML per lo scambio di messaggi strutturati tra applicazioni. Definito dallo standard W3C, utilizza WSDL per descrivere i servizi. Scelto nel progetto per compatibilità con il sistema Office Group esistente. 5.

Bibliografia

- [1] Oracle NetSuite, «What is ERP (Enterprise Resource Planning)?». [Online]. Disponibile su: <https://www.netsuite.com/portal/resource/articles/erp/what-is-erp.shtml>
- [2] «Scrum in Agile». [Online]. Disponibile su: <https://www.atlassian.com/it/agile/scrum/agile-vs-scrum>
- [3] Microsoft Corporation, «Azure DevOps Documentation». [Online]. Disponibile su: <https://learn.microsoft.com/en-us/azure/devops/>
- [4] DDEV Community, «DDEV Local Development Environment». [Online]. Disponibile su: <https://ddev.readthedocs.io/>
- [5] Docker Inc., «Docker Documentation». [Online]. Disponibile su: <https://docs.docker.com/>
- [6] SmartBear Software, «SoapUI Documentation». [Online]. Disponibile su: <https://www.soapui.org/docs/>
- [7] Postman Inc., «Postman Learning Center». [Online]. Disponibile su: <https://learning.postman.com/>
- [8] World Wide Web Consortium (W3C), «SOAP Version 1.2 Part 1: Messaging Framework (Second Edition)». [Online]. Disponibile su: <https://www.w3.org/TR/soap12/>
- [9] Microsoft Corporation, «.NET 6 Documentation». [Online]. Disponibile su: <https://learn.microsoft.com/en-us/dotnet/core/whats-new/dotnet-6>
- [10] Microsoft Corporation, «C# Documentation». [Online]. Disponibile su: <https://learn.microsoft.com/en-us/dotnet/csharp/>
- [11] Microsoft Corporation, «SQL Server 2022 Documentation». [Online]. Disponibile su: <https://learn.microsoft.com/en-us/sql/sql-server/>
- [12] Microsoft Corporation, «OAuth 2.0 Client Credentials Flow». [Online]. Disponibile su: <https://learn.microsoft.com/en-us/azure/active-directory/develop/v2-oauth2-client-creds-grant-flow>
- [13] «REST (Representational State Transfer)». [Online]. Disponibile su: <https://www.zerodivision.it/glossario/rest-representational-state-transfer/>
- [14] IBM Cloud Education, «Three-tier Architecture». [Online]. Disponibile su: <https://www.ibm.com/topics/three-tier-architecture>

- [15] Microsoft Corporation, «Transact-SQL Reference». [Online]. Disponibile su: <https://learn.microsoft.com/en-us/sql/t-sql/language-reference>
- [16] MDN Web Docs, «Cross-Origin Resource Sharing (CORS)». [Online]. Disponibile su: <https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS>
- [17] Microsoft Corporation, «Microsoft Graph API Documentation». [Online]. Disponibile su: <https://learn.microsoft.com/en-us/graph/>
- [18] D. Hardt, «RFC 6749: The OAuth 2.0 Authorization Framework», ottobre 2012, *Internet Engineering Task Force (IETF)*. [Online]. Disponibile su: <https://datatracker.ietf.org/doc/html/rfc6749>
- [19] Microsoft Corporation, «Azure Active Directory Authentication Documentation». [Online]. Disponibile su: <https://learn.microsoft.com/en-us/azure/active-directory/develop/>
- [20] HtmlAgilityPack Contributors, «HTML Agility Pack Documentation». [Online]. Disponibile su: <https://html-agility-pack.net/>
- [21] Microsoft Corporation, «Azure Portal Documentation». [Online]. Disponibile su: <https://portal.azure.com/>
- [22] GeeksforGeeks, «ACID Properties in Database Management Systems». [Online]. Disponibile su: <https://www.geeksforgeeks.org/acid-properties-in-dbms/>