Università degli Studi di Padova

Dipartimento di Matematica "Tullio Levi-Civita"

Corso di Laurea in Informatica

Kargo: Applicazione Web per la Logistica dello Scarico Merci

Tesi di Laurea

Relatore

Prof. Vardanega Tullio

Laureando

Suar Alberto

Anno Accademico 2025-2026

© Suar Alberto, Luglio 2026. Tutti i diritti riservati. Tesi di Laurea: "*Kargo: Applicazione Web per la Logistica dello Scarico Merci*", Università degli Studi di Padova, Dipartimento di Matematica "Tullio Levi-Civita", Corso di Laurea in Informatica.

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

Ringraziamenti

Prima di tutto, desidero ringraziare profondamente la mia famiglia: mia mamma Katia, mio papà Andrea e, *soprattutto*, mio fratello Gabriele, per l'immenso e costante supporto che mi hanno sempre dato durante tutto il mio percorso di laurea.

Vorrei poi porgere i miei ringraziamenti ai "Poti" e alle "Pote", che hanno reso indimenticabili questi tre anni di università, condividendo con me lo studio, l'ansia degli esami e innumerevoli risate.

Un pensiero speciale va a tutti i miei amici, che mi sono stati a fianco e mi hanno incoraggiato durante la stesura di questa tesi.

Infine, desidero ringraziare il mio relatore, il Professor Tullio Vardanega, per la disponibilità, il rigore e il prezioso aiuto che mi ha fornito durante la stesura di questo elaborato.

Padova, Luglio 2026 *Suar Alberto*

Sommario

Nel presente documento descrivo il lavoro che ho svolto durante il periodo di *stage*, dal 5 maggio 2026 al 1 luglio 2026, presso l'azienda Ergon Informatica S.r.l.
Sotto la supervisione del Prof. Tullio Vardanega e del tutor aziendale Thomas Garbui, ho perseguito come obiettivo principale la realizzazione di una *web application* per la gestione logistica delle prenotazioni di banchine di scarico per mezzi pesanti, sviluppata impiegando React e C#.
Ho suddiviso il documento in **4 capitoli**:

- **Capitolo 1: Dentro Ergon Informatica**. Descrivo la struttura organizzativa dell'azienda, le metodologie di lavoro adottate e la propensione dell'azienda per l'innovazione.

- **Capitolo 2: Lo *stage* come innovazione**. Analizzo il ruolo del tirocinio all'interno della filosofia aziendale di innovazione, confrontando le motivazioni alla base della proposta progettuale e la rispondenza del progetto alle esigenze di mercato.

- **Capitolo 3: Sviluppare Kargo**. Presento il progetto Kargo, illustrando il ciclo di vita del *software*, le tecnologie utilizzate, le decisioni progettuali, le funzionalità implementate nell'applicativo e i risultati conseguiti.

- **Capitolo 4: Valutazione retrospettiva**. Effettuo un'analisi critica dei risultati ottenuti, comparando le aspettative iniziali con i traguardi finali e misurando l'impatto dello *stage* sullo sviluppo delle mie competenze professionali e personali.

Convenzioni tipografiche

Al fine di rendere più chiara la lettura del documento, ho adottato le seguenti convenzioni tipografiche:

- Il *corsivo* evidenzia esclusivamente i termini in lingua straniera e i titoli di opere o documenti.
- Il **grassetto** enfatizza concetti chiave, definizioni rilevanti e richiama l'attenzione su elementi specifici del testo.
- Le immagini e le figure riportano una numerazione progressiva e una didascalia descrittiva che ne chiarisce il contesto. Salvo diversa indicazione, ho realizzato personalmente tutte le immagini, i grafici e le tabelle presenti nel documento.
- Il simbolo $_G$ in pedice contrassegna i termini definiti nel glossario in appendice. Alla prima occorrenza, affianco al termine una breve definizione contestuale; il simbolo $_G$ funge da rimando diretto alla definizione completa presente in appendice.

Indice

Elenco delle figure

Elenco delle tabelle

Capitolo 1

Dentro Ergon Informatica

Nel presente capitolo delineo il contesto operativo all'interno del quale ho svolto l'attività di tirocinio e analizzo la struttura aziendale, le metodologie di sviluppo in uso e la propensione dell'organizzazione verso l'innovazione tecnologica.

1.1 Profilo aziendale e dominio applicativo

Ergon Informatica S.r.l. è una società operante nel settore dell'*Information Technology (IT)*$_G$ con sede a Castelfranco Veneto, in provincia di Treviso. L'azienda, che ha avviato le proprie attività nel 1988 e vanta un organico di circa 60 collaboratori, si concentra sullo sviluppo di applicativi gestionali per il comparto alimentare. Un secondo ramo societario, Ergon Servizi S.r.l., affianca oggi l'impresa nella gestione dei processi amministrativi, logistici e commerciali. Questa specializzazione di settore ha definito le funzionalità centrali degli applicativi aziendali: il *software* deve infatti garantire il rispetto di rigidi vincoli operativi e normativi propri della filiera agroalimentare, come la gestione completa della tracciabilità dei lotti di produzione e la supervisione della catena logistica. All'interno di questo dominio, la clientela di Ergon si distribuisce su tutto il territorio nazionale, comprendendo sia piccole e medie imprese sia alcune grandi aziende di rilievo nel panorama commerciale del settore, operanti in ambito prettamente privato.

1.1.1 Ecosistema applicativo e servizi *IT*

Il prodotto principale dell'azienda è l'applicativo Ergdis, un *Enterprise Resource Planning (ERP)*$_G$ — ovvero un sistema informativo centralizzato che permette a un'azienda di pianificare, gestire e monitorare tutte le proprie risorse operative e processi aziendali. Il sistema adotta un'architettura modulare distribuita che poggia sul modello *Client-Server*$_G$ — un paradigma di rete in cui i compiti sono ripartiti tra elaboratori che forniscono servizi, i *server*, e terminali che li richiedono, i *client* — come illustra la Figura 1.1. L'integrazione logica tra i terminali periferici e il nodo centrale avviene tramite la *suite* di protocolli *Transmission Control Protocol/Internet Protocol (TCP/IP)*$_G$, lo standard universale per la trasmissione dei dati in rete, avvalendosi del *file system* Samba, un programma che permette la condivisione trasparente di *file* e stampanti tra sistemi operativi differenti.

L'architettura di Ergdis utilizza un approccio ibrido per la ripartizione dei carichi di lavoro:

- **Livello *client* (Presentazione):** operante in ambiente Windows e sviluppato in Visual Basic 6, gestisce le interfacce utente. A tale scopo, il livello si avvale dei componenti *OLE Control Extension (OCX)*$_G$, speciali librerie *software* modulari precompilate che l'ambiente Microsoft adotta per estendere le funzionalità visive o logiche delle interfacce, e del generatore di stampe Crystal Reports 8.0.

- **Livello *server* (Elaborazione e persistenza):** risiede in ambiente Unix ed esegue le procedure *Batch*$_G$ — ovvero l'esecuzione automatizzata e sequenziale di elaborazioni massive sui dati senza necessità di interazione umana. Gli sviluppatori implementano tali procedure in ESQL/C, un'estensione del linguaggio di programmazione C che permette di incorporare le interrogazioni al *database* direttamente all'interno del codice sorgente. Il sistema di gestione di basi di dati relazionali (*Relational Database Management System (RDBMS)*$_G$) Informix Dynamic Server 7.3 governa la

persistenza del dato, organizzando le informazioni in tabelle interconnesse da relazioni logiche.

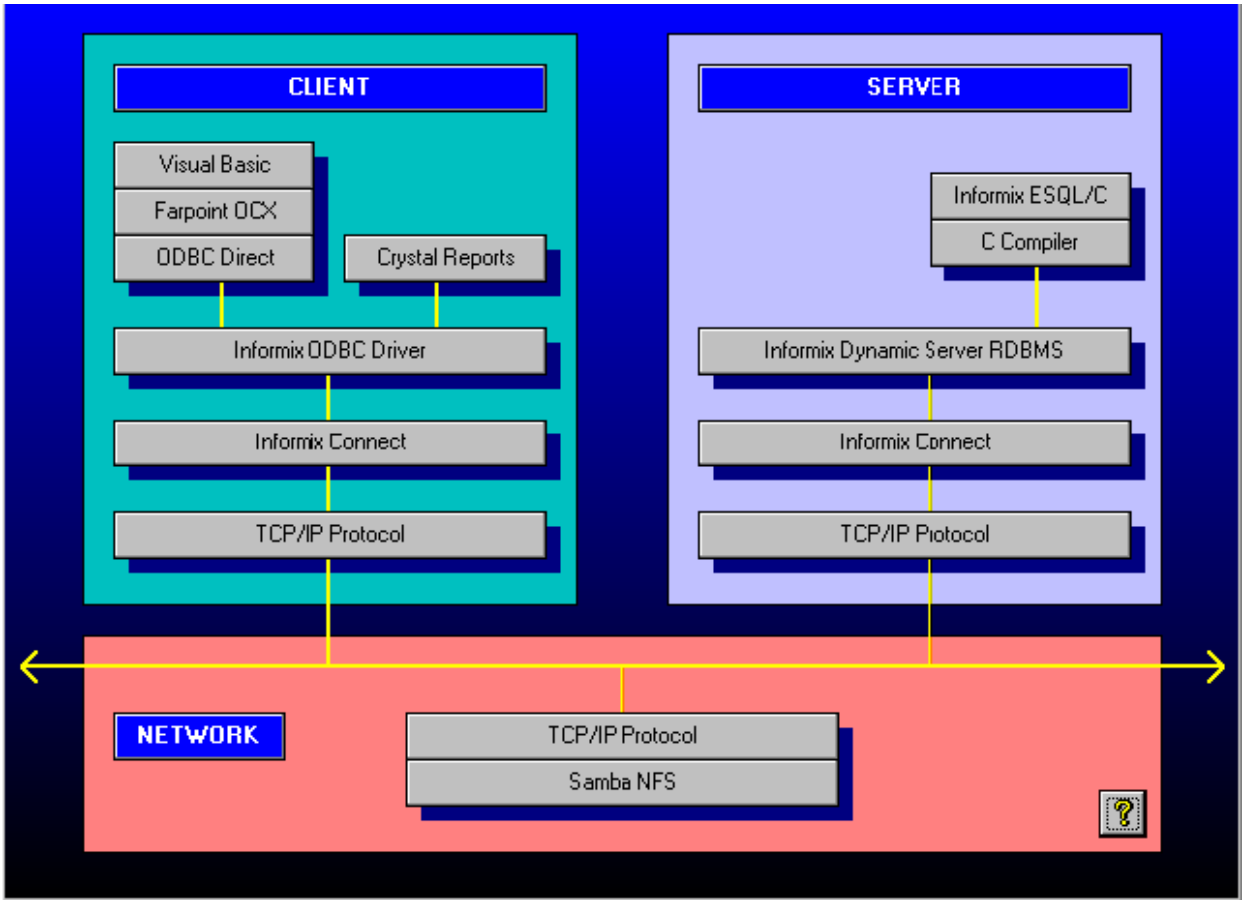

Figura 1.1: Architettura *client-server* dell'applicativo Ergdis.

Fonte: *Ergdis - Linee guida per lo Sviluppo*, documentazione interna aziendale

Tale disaccoppiamento logico permette a Ergdis di operare come un ecosistema di programmi indipendenti che orbitano attorno a un nucleo di contabilità generale e fatturazione, a cui si affiancano moduli dedicati alla gestione del magazzino, degli ordini verso clienti e fornitori, delle provvigioni e dei cespiti, come evidenzia la Figura 1.2. Un aspetto ingegneristico di rilievo, che la documentazione di sistema attesta, è l'interfacciamento a basso livello con l'infrastruttura *hardware* industriale dei clienti. Il sistema deve infatti comunicare direttamente con dispositivi fisici quali linee di confezionamento, lettori ottici e bilance industriali, richiedendo l'implementazione di protocolli specifici per acquisire i dati di produzione in tempo reale e garantire il corretto avanzamento delle operazioni di fabbrica.

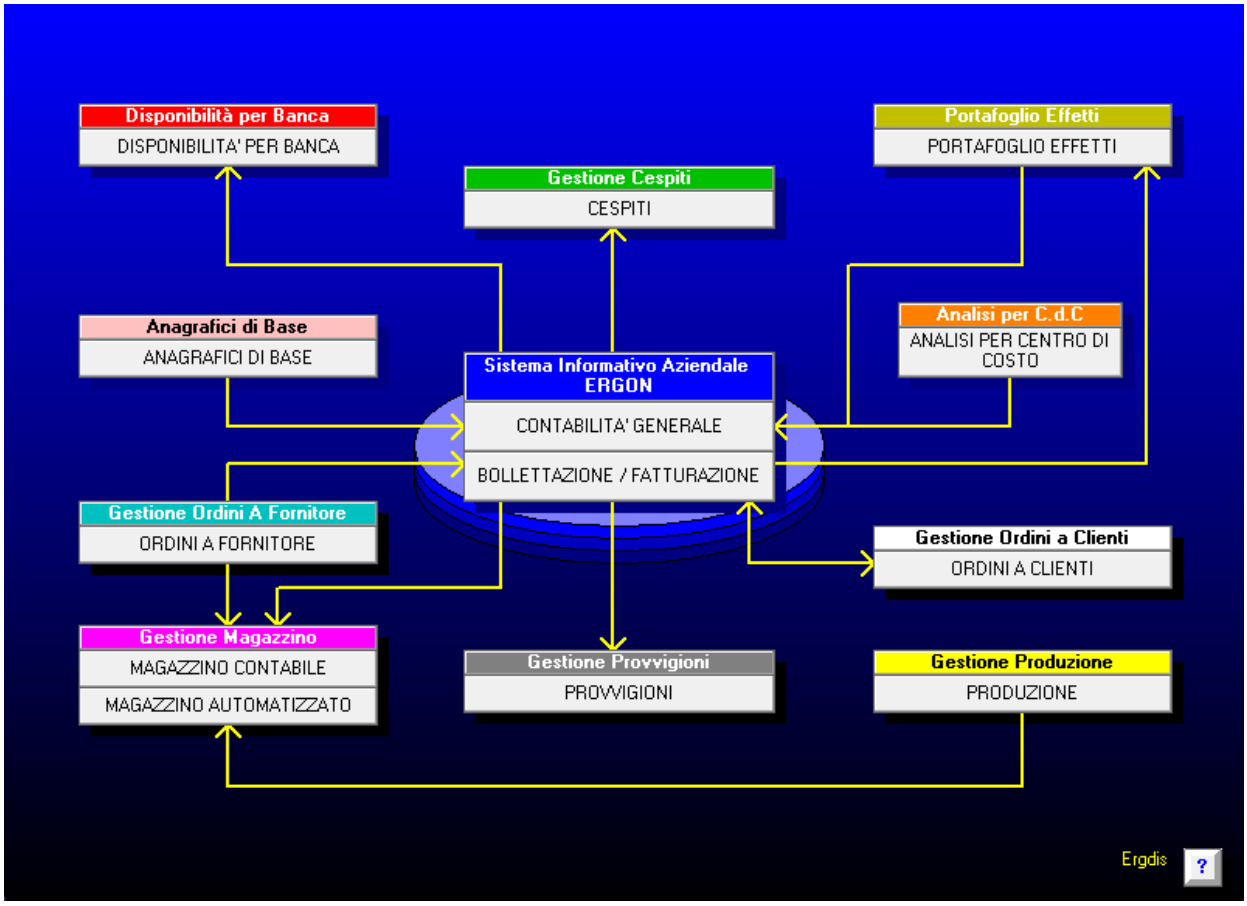

Figura 1.2: Moduli funzionali del sistema informativo Ergdis.

Fonte: *Ergdis - Linee guida per lo Sviluppo*, documentazione interna aziendale

A questa base si affiancano moduli *web-based* dedicati al *Customer Relationship Management (CRM)*$_G$ — strumenti informatici progettati per la tracciatura, l'analisi e la gestione di tutte le relazioni con i clienti — e ulteriori applicativi verticali. Tra questi spiccano Prodector, uno strumento di predizione degli ordini che impiega modelli di *Machine Learning*$_G$, ovvero algoritmi di intelligenza artificiale capaci di apprendere autonomamente dai dati storici per formulare previsioni, e un modulo di *Geomarketing*$_G$, che sfrutta l'analisi spaziale e geografica dei dati per guidare le decisioni commerciali. L'applicativo Ergvis completa l'offerta, introducendo la modellazione tridimensionale degli ambienti produttivi. In ambito infrastrutturale, l'analisi evidenzia l'uso di tecnologie di virtualizzazione VMware, che permettono di eseguire e isolare molteplici *server* virtuali su un unico calcolatore fisico, e l'implementazione di *policy* di sicurezza perimetrale mediante *Firewall*$_G$ — apparati di sicurezza che monitorano e

filtrano selettivamente il traffico di rete.

1.2 Processi e metodologie di sviluppo

Per ricostruire le metodologie operative in uso presso l'azienda, l'indagine preliminare si basa su interviste con il responsabile aziendale, sull'analisi della documentazione interna e sul confronto con i colleghi del reparto di sviluppo.

1.2.1 Organizzazione del *team* e rapporto con il cliente

Il modello operativo di Ergon si discosta nettamente dalle metodologie agili formali, come *Scrum*$_G$, un noto *framework* di lavoro che organizza lo sviluppo in brevi cicli iterativi e rigorosamente scanditi. L'azienda predilige un approccio incrementale, privo di formalismi e fortemente orientato al riuso del codice. Ne consegue l'assenza di cerimonie strutturate tipiche dei metodi agili, come le riunioni giornaliere di allineamento o le pianificazioni formali a inizio ciclo. L'introduzione di tali pratiche e di figure dedicate alla pura facilitazione dei processi, come lo *Scrum Master*, comporterebbe un costo organizzativo eccessivo per il numero limitato di risorse, considerata anche la semplicità e l'immediatezza della comunicazione aziendale, che il personale risolve spesso spostandosi fisicamente tra gli uffici.

Il responsabile tecnico centralizza la ripartizione del carico di lavoro: egli delinea i *team* di sviluppo — solitamente ristretti, in cui i programmatori collaborano direttamente con i sistemisti — assegnando le mansioni in base alle specifiche competenze di dominio che il prodotto richiede. Come conferma il *tutor*, questa organizzazione rappresenta un supporto fondamentale, in quanto solleva gli sviluppatori dall'onere della pianificazione organizzativa autonoma e permette loro di concentrarsi esclusivamente sulla stesura del codice. Inoltre, l'assenza di figure di intermediazione per la pura gestione dei progetti, come i *Project Manager*, fa sì che lo sviluppatore incaricato gestisca direttamente il rapporto con il cliente. Insieme al reparto commerciale, lo sviluppatore si assu-

me frequentemente anche la responsabilità della stesura e della definizione del preventivo economico.

1.2.2 Il ciclo di vita del *software*

Il ciclo di vita del *software* adotta un modello iterativo in cui l'azienda non dichiara mai del tutto conclusa l'attività di analisi dei requisiti. Questo approccio permette di assecondare le continue richieste di modifica o integrazione da parte del cliente sui prodotti già in uso. In questo contesto, gli sviluppatori accoppiano strettamente l'attività di progettazione dell'architettura logica alla codifica effettiva, eseguendole parallelamente, come mostra la Figura 1.3.

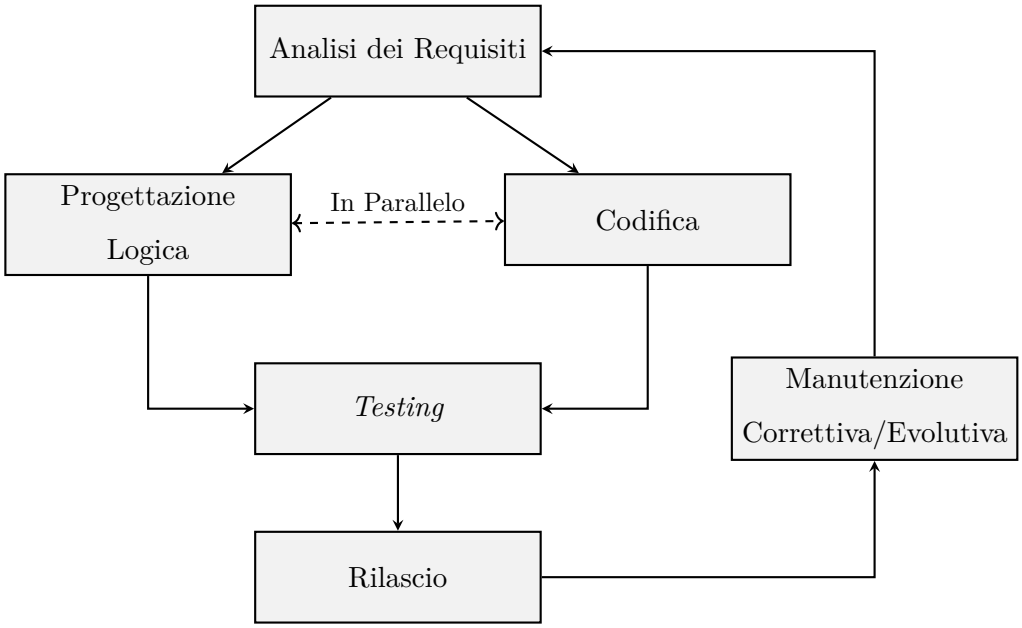

Figura 1.3: Rappresentazione schematica del ciclo di vita del *software* in Ergon.

Elaborazione originale dell'autore

L'adozione di uno *stack* tecnologico consolidato e la limitatezza delle risorse disponibili, rendono superflua la stesura di documenti tecnici preliminari. Il *team* omette lo studio formale di fattibilità data l'estrema familiarità con le tecnologie in uso. Parallelamente, una massiccia e strutturata presenza di commenti esplicativi all'interno del codice sorgente sostituisce la specifica tecnica di dettaglio; tali annotazioni giustificano le scelte di *design* e guidano gli interventi di manutenzione futuri.

1.2.3 Convenzioni di codifica e architettura logica

Coerentemente con l'approccio adottato in sede di analisi, in cui gli analisti scompongono i problemi complessi in unità logiche minori, anche la codifica segue una rigorosa logica procedurale dall'alto verso il basso, che assume il nome di *top-down*. Gli sviluppatori articolano i programmi in procedure principali che, all'aumentare della complessità, frammentano a loro volta in sotto-procedure specializzate. I singoli blocchi di codice si scambiano le informazioni necessarie in modo esplicito attraverso il passaggio di parametri, confinando il più possibile l'utilizzo di variabili globali accessibili dall'intero sistema.

L'attività di codifica non prevede l'adozione sistematica di *pattern* architetturali moderni o la rigida applicazione dei principi *SOLID*$_G$, un *set* di regole per favorire la creazione di *software* altamente disaccoppiato e manutenibile. I limiti fisici e strutturali del linguaggio storico e la fortissima dipendenza dalle logiche del dominio alimentare guidano tale scelta organizzativa: per gli sviluppatori risulta operativamente più vantaggioso strutturare il codice caso per caso, piuttosto che sforzarsi di piegarlo a schemi teorici predefiniti.

L'azienda impone tuttavia delle regole formali interne inderogabili per la stesura del codice:

- **Tipizzazione coerente:** ogni variabile deve possedere un tipo di dato strettamente conforme alla natura dell'informazione che essa contiene, limitando fortemente l'impiego del tipo generico `Variant`, il quale rallenta l'esecuzione e rende i controlli meno sicuri.

- **Dichiarazione esplicita:** lo sviluppatore deve obbligatoriamente dichiarare l'esistenza di ogni variabile all'inizio della procedura, prima di poterle assegnare un valore o utilizzarla.

- **Gestione degli identificatori:** Ergon vieta categoricamente di assegnare lo stesso nome a variabili locali appartenenti a procedure diverse, garantendo così l'assoluta assenza di ambiguità durante la lettura e la rielaborazione successiva del codice.

A livello lessicale, i programmatori devono rispettare le convenzioni di nomenclatura che la Tabella 1.1 illustra.

Elemento	**Convenzione**
Variabili e funzioni (in inglese)	*PascalCase*
Tabelle e *stored procedure*	`MACRO_CASE`
Recordset	prefisso `ds`
Interrogazioni ai dati (*query*)	prefisso `qdf`
Stringhe per istruzioni SQL	prefisso `sql`
Variabili collegate alle basi di dati	nome del campo della tabella

Tabella 1.1: Convenzioni di nomenclatura adottate nel codice e nel *database*.

Fonte: *Ergdis - Linee guida per lo Sviluppo*, documentazione interna aziendale

1.2.4 *Testing*, rilascio e manutenzione

Ergon non formalizza l'attività di collaudo attraverso stesure documentali, né si avvale di metodologie di esecuzione automatizzata dei *test* o di strumenti di *Code Coverage*$_G$ per calcolare con precisione matematica quanta parte del codice superi la verifica prima del rilascio. Gli sviluppatori garantiscono la validazione basandosi sull'esecuzione continua di *test* manuali di integrazione direttamente sull'interfaccia. L'obiettivo primario di questa prassi è intercettare le anomalie di comunicazione tra i vari componenti mentre il programmatore costruisce il *software*, prima di stratificarvi sopra nuove logiche. Data l'assegnazione individuale dei compiti — che delega la completa responsabilità di un modulo a un singolo programmatore — il reparto non istituisce procedure di *code review*, cioè di revisione incrociata del codice tra colleghi.

Il rilascio in produzione avviene in modalità incrementale. In assenza di piattaforme dedicate all'integrazione continua o al rilascio automatico in produzione (*Continuous Integration / Continuous Delivery (CI/CD)*$_G$), lo sviluppatore confeziona e installa manualmente i pacchetti di aggiornamento. Egli stesso stabilisce la prontezza del progetto nel momento in cui giudica le funzionalità conformi alle richieste del committente; organizza la consegna formale e

la correda con sessioni di installazione tecnica e di addestramento degli utenti finali.

L'onere della manutenzione, che comprende sia la correzione di difetti preesistenti sia lo sviluppo di nuove funzioni, ricade interamente sul programmatore che ha curato lo sviluppo originale. Le richieste di assistenza, che i clienti inoltrano al centralino o comunicano al tecnico in via diretta, spingono lo sviluppatore ad affinare l'analisi iniziale e a delineare l'intervento tecnico necessario, con la conseguente riapertura della stima economica dei lavori. Il responsabile tecnico stabilisce infine l'urgenza di esecuzione dei compiti, soppesando attentamente le scadenze, il *budget* che la dirigenza ha approvato, la ristrutturazione del codice che la modifica richiede e la disponibilità pratica della risorsa.

Questa impostazione operativa risponde all'esigenza primaria dell'azienda: poiché ogni singolo sviluppatore deve garantire supporto simultaneo a molteplici clienti, il reparto tara il ciclo produttivo per massimizzare la rapida consegna di risultati concreti. L'adozione di rigidi protocolli di validazione o di cicli di vita più burocratizzati imporrebbe un carico strutturale insostenibile per la necessaria flessibilità, penalizzando severamente i tempi di consegna che costituiscono il punto di forza dell'impresa.

1.3 Innovazione tecnologica e cultura aziendale

L'analisi nei paragrafi precedenti evidenzia un ecosistema applicativo fortemente radicato su uno *stack* tecnologico *legacy*, incentrato sull'impiego di Visual Basic 6, dei *database* Informix e dei componenti DevExpress. Tali tecnologie, sebbene garantiscano l'incrollabile stabilità e la solidità operativa che il comparto della produzione alimentare pretende, costituiscono al contempo un evidente e fisiologico debito tecnico che la dirigenza sta affrontando attivamente. L'approccio di Ergon all'innovazione, tuttavia, non si traduce nell'immediato stravolgimento dei sistemi esistenti, bensì in una lenta transizione incrementale e cautelativa, che la dirigenza soppesa costantemente in relazione ai costi aziendali, ai rischi sistemistici e al mantenimento ininterrotto dei servizi per la clientela.

1.3.1 Transizione architetturale e nuovi standard

L'aggiornamento degli strumenti di supporto allo sviluppo costituisce un chiaro indicatore di questo percorso di transizione. Fino a poco tempo fa, infatti, i programmatori gestivano la cronologia del codice sorgente in totale assenza di un *Version Control System (VCS)*$_G$, ovvero un registro strutturato per il controllo delle versioni e delle modifiche. I *developer* affidavano il tracciamento esclusivamente al salvataggio manuale di intere cartelle e archivi compressi: lo sviluppatore scaricava il progetto dal *server* principale sulla propria macchina, lavorava i *file* localmente e, a lavoro terminato, ne riversava la copia aggiornata nello spazio condiviso, sovrascrivendo la versione precedente. Solo nel corso dell'ultimo semestre l'azienda ha introdotto e imposto l'utilizzo di Git, uno degli strumenti di controllo di versione distribuito più diffusi sul mercato. Questo salto metodologico pone fondamenta solide per una collaborazione coordinata tra colleghi e per una sicura tracciabilità storica delle alterazioni dei *file*.

Sul piano puramente ingegneristico, Ergon ha parallelamente avviato uno studio di rifattorizzazione dell'architettura. L'obiettivo strategico di lungo periodo è la migrazione per gradi del nucleo gestionale Ergdis dall'obsoleto Visual Basic 6 al moderno *framework* .NET, avvalendosi del linguaggio C# e di Entity Framework, uno strumento di mappatura che trasforma le tabelle relazionali dei dati in oggetti informatici facili da interrogare nel codice. Questa rotta evolutiva include l'esplorazione dei servizi che le infrastrutture *cloud* di Microsoft Azure offrono; l'azienda impiega tali servizi sia per esternalizzare l'infrastruttura di distribuzione dei programmi, sia per condurre sperimentazioni con l'intelligenza artificiale, in particolare con i *Large Language Model (LLM)*$_G$, modelli logico-linguistici in grado di decodificare e generare testo simulando l'elaborazione umana. L'insieme di queste iniziative testimonia la concreta volontà aziendale di emanciparsi dalla subordinazione ai vecchi standard e di accorciare la distanza tecnologica che la separa dall'offerta del mercato contemporaneo.

1.3.2 Il ruolo strategico dei tirocini

All'interno di questo misurato scacchiere di modernizzazione, Ergon assegna un ruolo tattico ai progetti di tirocinio formativo. La dirigenza valorizza infatti le collaborazioni universitarie come veri e propri laboratori di *Research and Development (R&D)*$_G$, il dipartimento che gestisce la ricerca e lo sviluppo tecnologico. Invece di azzardare l'introduzione di strumenti moderni o librerie pionieristiche direttamente sull'ossatura del prodotto gestionale principale — mossa che potrebbe introdurre instabilità critiche nei sistemi in uso ai clienti — i responsabili preferiscono delegare l'esplorazione di questi nuovi *framework* agli studenti stagisti.

Questo espediente organizzativo permette alla direzione tecnica di ricavare metriche empiriche affidabili, misurare la curva di apprendimento, le prestazioni reali e la difficoltà di integrazione delle nuove tecnologie nell'ecosistema aziendale. I tirocini agiscono perciò da testa di ponte: la costruzione di piccoli applicativi laterali, che assumono il nome di *Proof of Concept (PoC)*$_G$ — ossia prototipi funzionanti ideati esclusivamente per dimostrare e validare la fattibilità ingegneristica di un'idea — consente di testare i moderni stili di programmazione limitando il rischio d'impresa e tenendo le sperimentazioni rigorosamente isolate dal *software* storico di contabilità.

Capitolo 2

Lo *stage* come innovazione

Nel presente capitolo approfondisco le motivazioni strategiche alla base della proposta di tirocinio e la colloco nel più ampio quadro della visione aziendale di Ergon Informatica. Attraverso lo studio degli obiettivi formativi e dei vincoli progettuali, illustro come lo stage *si inserisca nella strategia aziendale di innovazione e di miglioramento continuo dei processi. Le considerazioni derivano principalmente dalle interviste con il* tutor *aziendale e dalle osservazioni dirette sul campo durante il periodo di permanenza in azienda.*

2.1 Il valore dello *stage* in Ergon Informatica

Come espone la Sezione 1.3.2, Ergon Informatica considera il tirocinio uno degli strumenti principali per l'esplorazione tecnologica. Per alimentare costantemente questo ecosistema di sperimentazione, l'organizzazione ha strutturato una solida rete di collaborazioni con le principali istituzioni formative del territorio e accoglie annualmente oltre una decina di tirocinanti.

2.1.1 Collaborazioni con le istituzioni formative

L'azienda seleziona candidati da provenienze volutamente eterogenee, che spaziano dai percorsi accademici tradizionali — come l'Università degli Studi di Padova e l'Università Ca' Foscari di Venezia — fino a iter maggiormente professionalizzanti, come i vari Istituti Tecnici Superiori (ITS) della regione.

Questa diversificazione consente all'azienda di individuare con continuità nuovi talenti e, aspetto ancor più rilevante, di importare approcci mentali e operativi differenti. Come illustra la Tabella 2.1, i profili universitari apportano un maggiore rigore ingegneristico e metodologico, mentre i profili tecnici introducono una spiccata propensione per l'operatività immediata.

Profilo del Tirocinante	**Contributo Metodologico e Operativo**
Percorsi Universitari	Rigore ingegneristico, approfondimento teorico e approccio strutturato
Istituti Tecnici Superiori	Immediatezza esecutiva, pragmatismo operativo e orientamento pratico

Tabella 2.1: Diversificazione degli approcci metodologici apportati dai tirocinanti.

Elaborazione originale dell'autore

L'inserimento di queste risorse consente inoltre all'organizzazione di osservare indirettamente l'evoluzione del panorama tecnologico e formativo esterno. Il confronto con studenti di diversa estrazione formativa permette ai responsabili di raccogliere indicazioni sulle tecnologie che i percorsi di studio diffondono maggiormente, sugli strumenti emergenti e sugli approcci metodologici più recenti. In tale prospettiva, i tirocinanti diventano il collegamento tra il contesto accademico e quello industriale.

2.1.2 Integrazione del tirocinante nel contesto aziendale

I dati dell'indagine mostrano come Ergon progetti le attività di tirocinio in modo da escludere mansioni puramente esecutive o a basso valore formativo. I responsabili concepiscono le iniziative per coniugare le esigenze concrete dell'organizzazione con gli interessi e il percorso del candidato; tale strategia mira a generare un coinvolgimento attivo e a favorire la validazione di soluzioni dal reale potenziale d'impatto.

Questa dinamica innesca uno scambio bidirezionale di competenze all'interno dell'azienda: i tirocinanti assimilano la profonda conoscenza del dominio applicativo e delle logiche commerciali direttamente dai colleghi esperti, mentre

le figure *senior* esplorano le tecnologie emergenti e i paradigmi moderni che i più giovani introducono. In questo scenario, il tirocinante non agisce da semplice esecutore, bensì contribuisce attivamente allo studio di strumenti e approcci alternativi; tale contributo offre all'organizzazione l'opportunità di osservare nuove modalità operative e di valutarne l'eventuale adozione futura.

2.2 Ciclo di vita del tirocinio

Affinché l'apporto innovativo degli stagisti si traduca in un *asset* tangibile e non in una dispersione di energie, Ergon ha istituzionalizzato un modello operativo rigoroso: l'azienda inserisce ogni progetto formativo in un ciclo di vita continuo, la cui architettura prevede un chiaro "prima" e un altrettanto definito "dopo" rispetto all'orizzonte temporale della permanenza dello studente.

2.2.1 La genesi del progetto

Il ciclo di vita di un progetto di *stage* inizia molto prima dell'arrivo del candidato. La direzione definisce le iniziative a partire da problematiche operative concrete, che scaturiscono da due fonti primarie:

- **Impulsi di mercato:** specifiche richieste della clientela per funzionalità moderne che il gestionale storico non riesce a erogare agilmente, come ad esempio cruscotti *web* interattivi, interfacce *mobile* per gli operatori logistici o portali di comunicazione *Business-to-Business (B2B)*$_G$ — ovvero sistemi concepiti per l'interazione commerciale diretta tra aziende, senza l'intermediazione del consumatore finale.

- **Esigenze infrastrutturali interne:** colli di bottiglia architettonici che la dirigenza tecnica rileva e che richiedono un'indagine preliminare su strumenti più efficienti — quali moderni *Object-Relational Mapping (ORM)*$_G$, ovvero sistemi che trasformano i dati relazionali in oggetti informatici, o architetture *Representational State Transfer (RESTful)*$_G$, paradigmi di rete che permettono a sistemi diversi di scambiarsi informazioni in modo standardizzato — prima della loro adozione definitiva.

Quando emerge una di queste esigenze, il *team* tecnico delimita il perimetro funzionale e individua uno *stack* tecnologico coerente con gli obiettivi esplorativi del progetto. Il gruppo di lavoro traduce così il problema operativo in un'attività di tirocinio delimitata e verificabile, con l'obiettivo di produrre sia un risultato concreto sia elementi utili alla valutazione di possibili evoluzioni tecnologiche.

2.2.2 Il processo di validazione tecnica

Secondo il modello organizzativo che il *tutor* aziendale ha illustrato, la conclusione formale del tirocinio avvia generalmente una fase di consolidamento dei risultati ottenuti. L'azienda persegue l'obiettivo di produrre un modulo sufficientemente autonomo da permetterne una valutazione in termini tecnici e organizzativi; anche nei casi in cui il livello di completezza non consenta un impiego immediato, la dirigenza conserva il lavoro svolto e lo considera una base di riferimento per sviluppi successivi.

Al termine dello *stage*, gli sviluppatori *senior* sottopongono il codice sorgente, le scelte architetturali effettuate, la documentazione redatta e le pratiche di collaudo a una rigorosa procedura di *code review*. I responsabili tecnici non si limitano a verificare il funzionamento dell'interfaccia, ma effettuano un *audit* approfondito per valutare la manutenibilità del codice, la sicurezza delle *Application Programming Interface (API)*$_G$ — interfacce di programmazione che consentono a sistemi *software* differenti di dialogare tra loro — e la reale scalabilità delle tecnologie che il tirocinante propone.

2.2.3 Integrazione e sviluppo iterativo

A seguito del processo di validazione, il codice generato confluisce nel patrimonio informativo aziendale attraverso una di due possibili traiettorie di integrazione, a seconda della maturità del prodotto e della complessità del dominio in analisi:

1. **Messa in produzione diretta:** Se l'applicativo soddisfa i rigidi requisiti aziendali di affidabilità e lo *stack* tecnologico in uso dimostra la robustezza necessaria, il *software* entra in una fase di rifinitura interna. Gli sviluppatori *senior* collegano il modulo sperimentale ai sistemi di persistenza

di produzione e procedono con il rilascio progressivo ai clienti. In questo scenario, il tirocinio fornisce all'azienda un prodotto commerciale "chiavi in mano".

2. **Messa in detenzione e sviluppo iterativo:** Qualora il progetto affronti un dominio applicativo troppo vasto perché lo sviluppo possa esaurirlo nel tempo a disposizione, o se la tecnologia investigata richiede ulteriori validazioni architetturali, il lavoro dello stagista assume il ruolo di versione di riferimento. La dirigenza tecnica congela tale *baseline* e la riassegna successivamente come punto di inizio per un tirocinante futuro. Questo approccio a "staffetta" permette di aggredire problemi complessi attraverso la loro scomposizione in più cicli formativi, in modo da generare iterativamente valore incrementale.

In entrambi gli scenari, il tirocinio arricchisce il patrimonio tecnico e organizzativo dell'azienda attraverso la produzione di componenti o conoscenze riutilizzabili, e offre al contempo elementi utili per orientare le future decisioni sull'evoluzione tecnologica del sistema informativo. La Figura 2.1 riassume visivamente l'intero ciclo di vita esplorativo, dalla raccolta delle necessità iniziali fino alla diramazione conclusiva tra produzione e sviluppo iterativo.

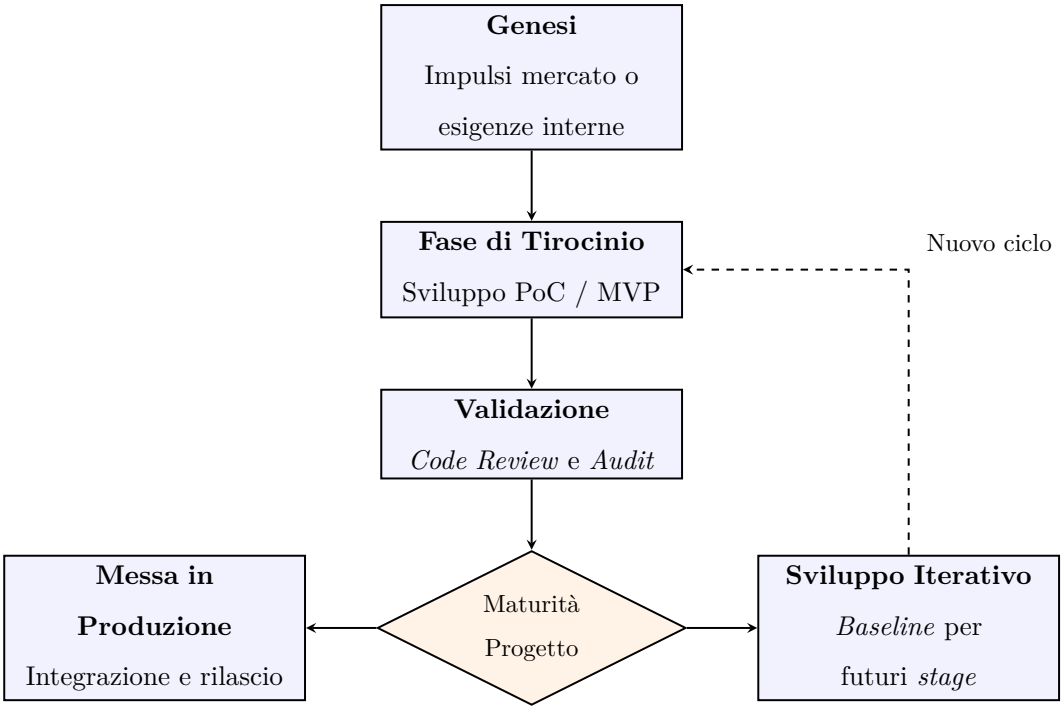

Figura 2.1: Rappresentazione schematica del ciclo di vita dei progetti in Ergon.

Elaborazione originale dell'autore

2.3 Il dominio applicativo del progetto Kargo

In aderenza alle logiche di assegnazione che le sezioni precedenti descrivono, il mio progetto di tirocinio — denominato "Kargo" — trae origine da una specifica esigenza di mercato. Un cliente di Ergon Informatica, che opera nel settore della distribuzione di bevande, ha manifestato la necessità di risolvere una forte criticità inerente alla logistica *inbound*: la congestione delle banchine di scarico.

Nella gestione quotidiana dei magazzini, l'assenza di programmazione negli arrivi dei vettori e la simultaneità delle consegne generano inefficienze a catena. I magazzinieri subiscono picchi di lavoro imprevedibili e accumulano ritardi nello stoccaggio, mentre i corrieri affrontano lunghi periodi di attesa nei piazzali prima di poter eseguire le operazioni di scarico, con un inevitabile aggravio dei costi operativi.

La dirigenza, che aveva ricevuto l'incarico di fornire una soluzione, aveva inizialmente pianificato uno sviluppo interno tradizionale. Tuttavia, poiché i responsabili tecnici hanno individuato in questa specifica problematica le dimen-

sioni ideali per uno *stage* e hanno confermato la necessità di esplorare nuove tecnologie *web*, l'azienda ha deciso di convertire la commessa in un progetto di tirocinio esplorativo.

2.3.1 Obiettivi e vincoli del progetto Kargo

In sede di confronto diretto con il *tutor* aziendale, ho stabilito congiuntamente il perimetro operativo del progetto Kargo, definendo obiettivi funzionali chiari e quantificabili e fissando precisi vincoli architetturali strategici.

L'obiettivo funzionale aziendale prevede lo sviluppo di un'applicazione *web* che digitalizzi e automatizzi la prenotazione degli *slot* temporali per lo scarico merci. Il sistema consente ai corrieri di monitorare le disponibilità delle banchine in tempo reale, di prenotare un appuntamento in sicurezza e offre all'amministrazione uno strumento di pianificazione ordinata. Per misurare il raggiungimento del risultato, Ergon ha fissato l'aspettativa di ricevere, entro le 300 ore previste dal tirocinio, un *Minimum Viable Product (MVP)*$_G$ — ovvero una versione iniziale del *software* limitata alle funzionalità essenziali per completare un intero ciclo logistico (dalla registrazione del corriere alla conferma dello *slot*) e per raccogliere i riscontri dei committenti — pronto per la successiva rifinitura e presentazione al cliente.

I vincoli tecnologici e architetturali che Ergon ha imposto risultano altrettanto rigorosi. L'azienda ha richiesto espressamente l'utilizzo del *framework* React per lo sviluppo del livello di presentazione (*frontend*), una tecnologia inedita rispetto agli standard interni. Inoltre, la direzione ha imposto un vincolo di stretta modularità architetturale: il progetto richiede un sistema agilmente integrabile, in un secondo momento, con il gestionale *legacy* proprietario che la Sezione 1.1.1 descrive. Questo requisito impone un approccio disaccoppiato, basato sullo sviluppo di API in ambiente .NET per il livello logico. La Figura 2.2 illustra questa netta separazione strutturale, evidenziando il perimetro operativo che l'azienda ha assegnato al progetto Kargo e le modalità di comunicazione con il preesistente livello di persistenza.

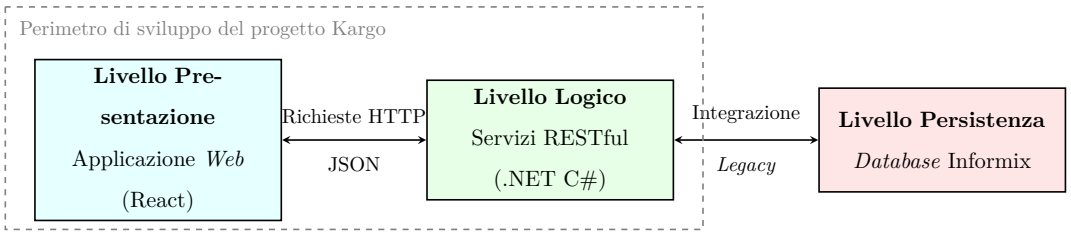

Figura 2.2: Vincoli di modularità architetturale imposti per il progetto Kargo.

Elaborazione originale dell'autore

2.4 Motivazioni e traguardi personali

Durante la fase di selezione delle proposte di *stage*, la mia candidatura iniziale si orientava in realtà verso un differente progetto interno, che prevedeva lo sviluppo di un sistema per il controllo degli accessi aziendali attraverso la tecnologia .NET MAUI. Tuttavia, a seguito di un colloquio orientativo con la direzione delle Risorse Umane, l'azienda mi ha proposto il progetto Kargo e ne ha illustrato lo *stack* tecnologico, le problematiche operative e gli obiettivi. Ho quindi scelto di orientare il mio percorso formativo verso quest'ultima proposta per motivazioni di natura sia tecnologica sia metodologica.

In primo luogo, dal punto di vista tecnico, Kargo rappresentava l'unica iniziativa a proporre lo sviluppo di interfacce tramite React. Rispetto alle alternative basate su *framework* maggiormente legati all'ecosistema *desktop* o *mobile* ibrido, l'opportunità di confrontarmi con lo sviluppo *web full-stack* moderno ha costituito per me un incentivo professionale determinante.

In secondo luogo, dal punto di vista metodologico, la natura esplorativa del progetto si allineava perfettamente con i miei obiettivi accademici. Durante il percorso universitario, ed in particolare nel corso di Ingegneria del Software, ho sviluppato un forte interesse per le tematiche relative al *System Design*$_G$ — il processo di definizione dell'architettura, dei componenti, delle interfacce e dei dati necessari per soddisfare requisiti specifici — e all'architettura del *software*. Un progetto che nasce al di fuori dei rigidi binari della produzione standard offriva un maggiore margine di manovra: garantiva la libertà necessaria per progettare un sistema *ex novo* e mi permetteva di applicare con rigore

i *pattern* architetturali di riferimento, analizzando criticamente le mie scelte ingegneristiche in un ambiente protetto.

A fronte di queste premesse, ho stabilito dei traguardi personali oggettivi, che si declinano nei seguenti punti:

- **Padronanza tecnologica e metodologica:** raggiungere la piena autonomia operativa nello sviluppo di applicazioni basate su C# e React — traguardo che la consegna del MVP nei tempi stabiliti comprova esplicitamente — e sviluppare parallelamente un metodo di studio efficiente per assimilare rapidamente nuovi linguaggi di programmazione.
- **Architettura e *problem solving*:** progettare un'architettura *software* applicando consapevolmente i principi di *system design*, che ottenga la validazione positiva del *team* tecnico durante le procedure di *code review*.
- **Gestione del ciclo di vita del *software*:** condurre operativamente tutte le principali fasi di produzione per tracciare il passaggio dalla progettazione iniziale alla validazione finale.

Mi sono posto l'obiettivo di verificare quanto le conoscenze teoriche del corso di laurea fossero effettivamente applicabili in un contesto industriale, in modo da consolidarle attraverso lo svolgimento diretto delle attività progettuali. I traguardi che questa sezione definisce costituiranno quindi il riferimento metrico con cui, nel Capitolo 4, valuterò oggettivamente i risultati finali del tirocinio.

Capitolo 3

Sviluppare Kargo

Il presente capitolo documenta le attività operative del progetto Kargo, illustra il metodo di lavoro scelto e l'evoluzione che porta alla realizzazione dell'applicativo. Attraverso l'analisi del ciclo di vita del software e delle principali sfide progettuali, descrivo il percorso che collega i requisiti iniziali alla validazione finale del sistema.

3.1 Metodologia di lavoro e interazione

Per garantire il corretto avanzamento del progetto all'interno del limitato orizzonte temporale dello *stage*, ho definito con il *tutor* aziendale un perimetro metodologico chiaro, stabilendo le regole di comunicazione e i criteri di validazione fin dal primo giorno di attività.

3.1.1 Modello di interazione aziendale e versionamento

In qualità di unico sviluppatore per il progetto Kargo, ho instaurato con il *tutor* — che ricopriva il ruolo di *stakeholder* di riferimento — una comunicazione fluida e continua, che la vicinanza fisica dei nostri uffici ha costantemente agevolato. Abbiamo optato per un approccio interattivo privo di scadenze formali rigide, prediligendo allineamenti mirati al bisogno.

Di fronte a ostacoli tecnici o dubbi implementativi, evitavo di presentare semplicemente l'anomalia e sottoponevo al *tutor* una procedura che si articola in tre passaggi:

1. Dimostrazione della comprensione della problematica.
2. Formulazione di una o più ipotesi di risoluzione, che lo studio teorico delle tecnologie in esame giustificava pienamente.
3. Applicazione concreta della soluzione all'interno del codice per dimostrarne il funzionamento.

A seguito di questa dimostrazione, il *tutor* mi forniva i propri riscontri. Le direttive ricevute differivano sensibilmente tra *frontend* e *backend*: il livello di presentazione subiva controlli estetici e funzionali stringenti per soddisfare i requisiti visivi del committente finale, mentre sul livello logico ho beneficiato di ampia libertà organizzativa e decisionale.

Sul piano del tracciamento del codice sorgente, l'azienda mi ha richiesto di mantenere l'approccio di storicizzazione manuale che la Sezione 1.3.1 descrive. Nonostante la mia familiarità con i moderni sistemi di versionamento come Git, l'assenza di procedure interne specifiche per queste piattaforme ha indotto il *tutor* a optare per il salvataggio tramite copie di *backup*. Ho quindi separato fisicamente il progetto in tre *directory* autonome: una cartella per il *backend* sviluppato in ambiente Visual Studio con .NET *Web* API, una per il *frontend* che ho generato tramite Vite per l'ecosistema React, e una terza per la documentazione tecnica. Sebbene avessi evidenziato alcune criticità in termini di manutenibilità, questo vincolo ha comportato in alcune occasioni un impiego di tempo operativo aggiuntivo per la ristrutturazione e il riordino dei *file*, e ha rallentato il flusso di sviluppo produttivo.

3.1.2 Metodologia di sviluppo personale e tracciamento

In assenza di piattaforme digitali per la gestione dei compiti — quali *board Kanban* o *software* di *ticketing* — e in assenza di una struttura di *Project Management*$_G$, ovvero la disciplina che organizza metodologicamente l'avvio, la

pianificazione e l'esecuzione di progetti aziendali, ho ideato un sistema organizzativo personale che impiega il supporto cartaceo per governare i cicli di sviluppo quotidiani.

Ho scelto il supporto analogico per la sua capacità di minimizzare il carico cognitivo associato all'uso di *software* di gestione esterna, una scelta che mi ha permesso di mantenere elevata la concentrazione sulla logica applicativa. Questa metodologia mi ha consentito di riflettere sulle regole di dominio prima della loro implementazione: la stesura a mano mi ha obbligato a sintetizzare le regole di *business* prima di tradurle nel codice, al fine di assicurare che ogni riga scritta risultasse funzionale a un obiettivo di dominio chiaro.

Per compensare la mancanza di scadenze aziendali formali e soddisfare i requisiti di tracciabilità metodologica, ogni giornata lavorativa iniziava con la stesura di un elenco di obiettivi, in modo da separare rigidamente le mansioni obbligatorie, necessarie per il proseguimento del flusso primario, da quelle opzionali. Per implementare ogni singola funzionalità, ho impiegato specifici strumenti di analisi e verifica, quali diagrammi *Unified Modeling Language (UML)*$_G$ — il linguaggio standardizzato per la rappresentazione grafica e la progettazione del *software* — per la modellazione, e interfacce di *test Hypertext Transfer Protocol (HTTP)*$_G$ — il protocollo di rete alla base del trasferimento delle informazioni sul *web* — per il collaudo. Per considerare un incarico effettivamente concluso, ho definito tre criteri di completamento, che ho monitorato costantemente per ogni singola attività:

- **Completamento funzionale:** l'applicativo doveva superare con successo l'operazione che il requisito richiedeva.
- **Documentazione contestuale:** il codice sorgente doveva presentare commenti esplicativi in ogni nodo logico del flusso, per fungere da guida alla lettura per il *tutor* e per i futuri sviluppatori.
- **Approvazione tecnica:** il componente doveva superare la sessione di *review* visiva con il responsabile aziendale, per certificare la conformità agli standard interni.

Come illustra la Figura 3.1, il supporto cartaceo ha permesso di tracciare rapidamente le logiche di dominio durante l'attività di analisi, catturando le relazioni tra entità e i flussi operativi prima di formalizzarli in diagrammi digitali. La Figura 3.2 mostra invece il formato adottato per il tracciamento giornaliero delle attività, con la distinzione visiva tra mansioni obbligatorie e opzionali e l'indicazione del loro stato di completamento attraverso la notazione *C-V-R* (Completato, Verificato, Revisionato), che mi ha permesso di monitorare sistematicamente lo stato di avanzamento e la qualità di ciascun incremento.

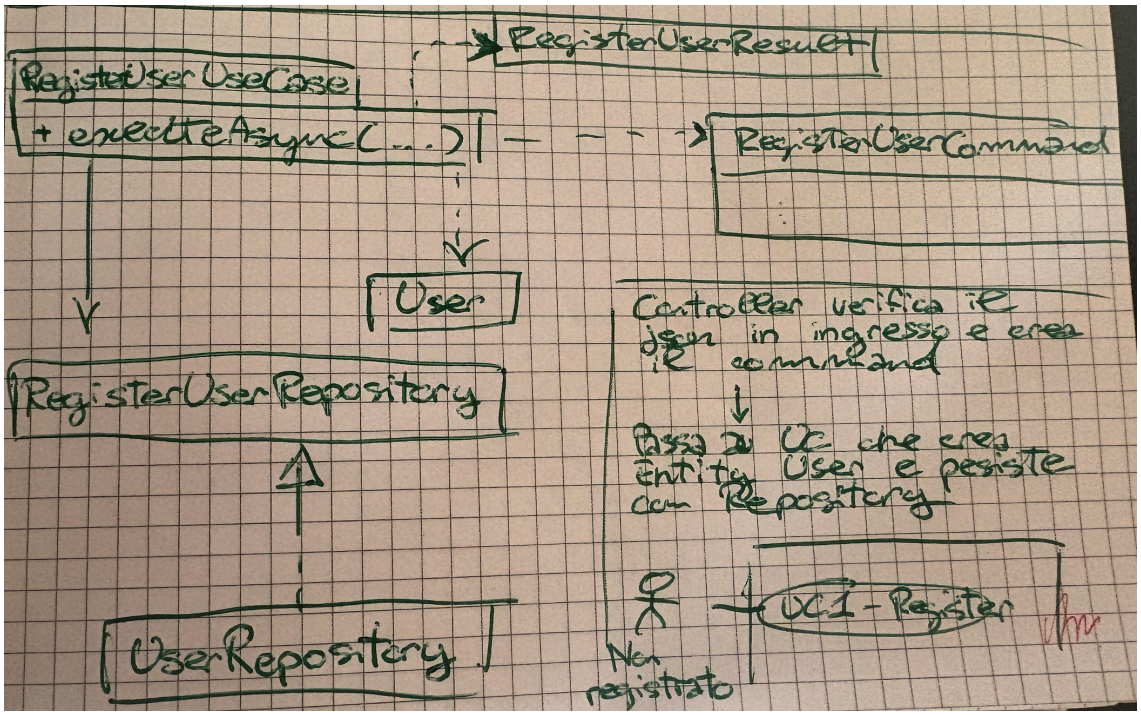

Figura 3.1: Appunti cartacei relativi all'attività di analisi del dominio applicativo e modellazione preliminare.

Elaborazione originale dell'autore

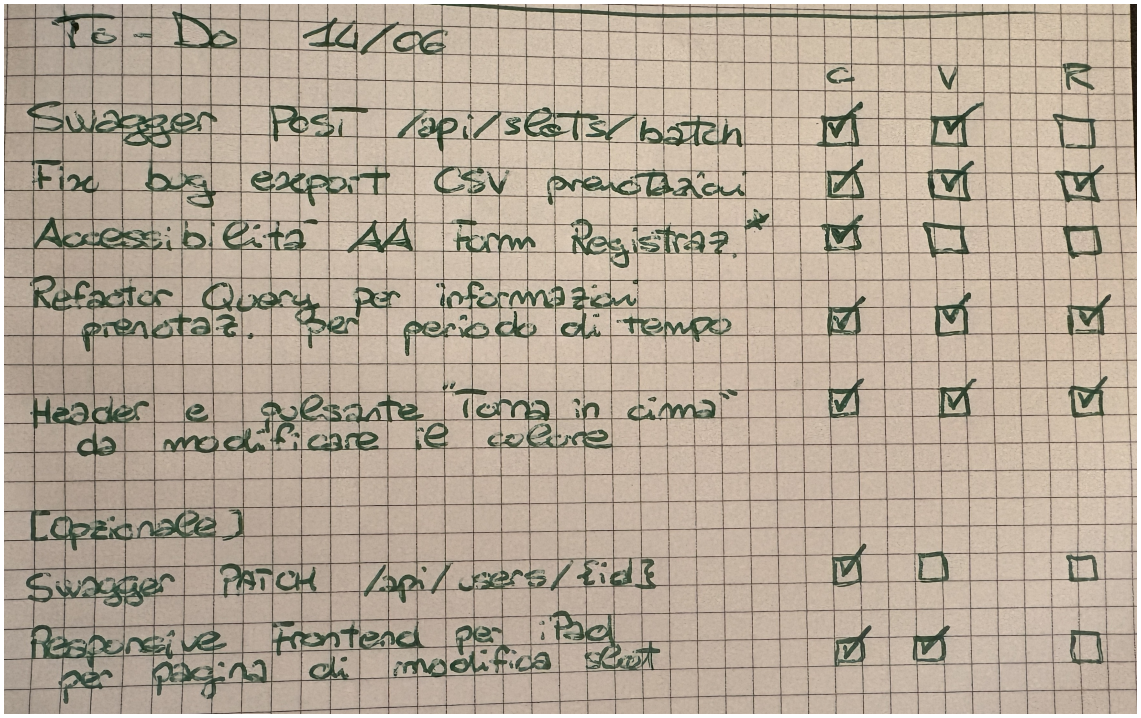

Figura 3.2: Sistema di tracciamento giornaliero basato su criteri di completamento, verifica e revisione.

Elaborazione originale dell'autore

La natura esplorativa del progetto ha richiesto inoltre una costante attività di auto-formazione. Ho supportato ogni attività di implementazione mediante la consultazione sistematica delle documentazioni ufficiali delle tecnologie in uso: la guida di riferimento di React e il portale di Microsoft Learn per l'ecosistema .NET. Questa consultazione mi ha permesso di comprendere le *best practice* di ogni *framework* e applicarle correttamente nel contesto aziendale.

3.2 Analisi dei requisiti e prototipo

Prima di avviare lo sviluppo vero e proprio, il *tutor* ha formalizzato i vincoli architetturali che la Sezione 2.3.1 espone e mi ha introdotto al progetto con il suo nome provvisorio: "*Web Service* Prenotazione Scarichi". L'obiettivo primario consisteva nel fornire ai corrieri una piattaforma per registrarsi e prenotare le baie dei depositi, centralizzando il controllo delle richieste.

In assenza di linee guida aziendali rigide per la stesura della documentazione preliminare, ho applicato i principi del corso di Ingegneria del *Software* per con-

durre un'attività di analisi dei requisiti strutturata. Ho adottato un approccio gerarchico: sono partito dalla definizione dei quattro attori principali di sistema (Utente non registrato, Utente non autenticato, Utente autenticato e Amministratore), per poi isolare 23 casi d'uso principali. Per ognuno di essi ho redatto un livello di dettaglio che prevede la descrizione dei singoli passaggi, comprese le relazioni di inclusione ed estensione e le catene operative che ne scaturiscono. Questo metodo ha consentito di granulare i requisiti in maniera precisa, rendendoli direttamente implementabili e verificabili attraverso le *checklist* giornaliere che la Sezione 3.1.2 descrive.

Ho redatto un documento contenente la descrizione del dominio e ho tracciato i diagrammi dei casi d'uso tramite PlantUML — uno strumento che genera diagrammi UML a partire da una sintassi testuale — per rappresentare visivamente ogni scenario di interazione. Successivamente, ho mappato i requisiti e li ho consolidati all'interno di apposite tabelle di classificazione. La Figura 3.3 e la Figura 3.4 mostrano un estratto del lavoro di analisi.

UC9 - Creazione Prenotazione

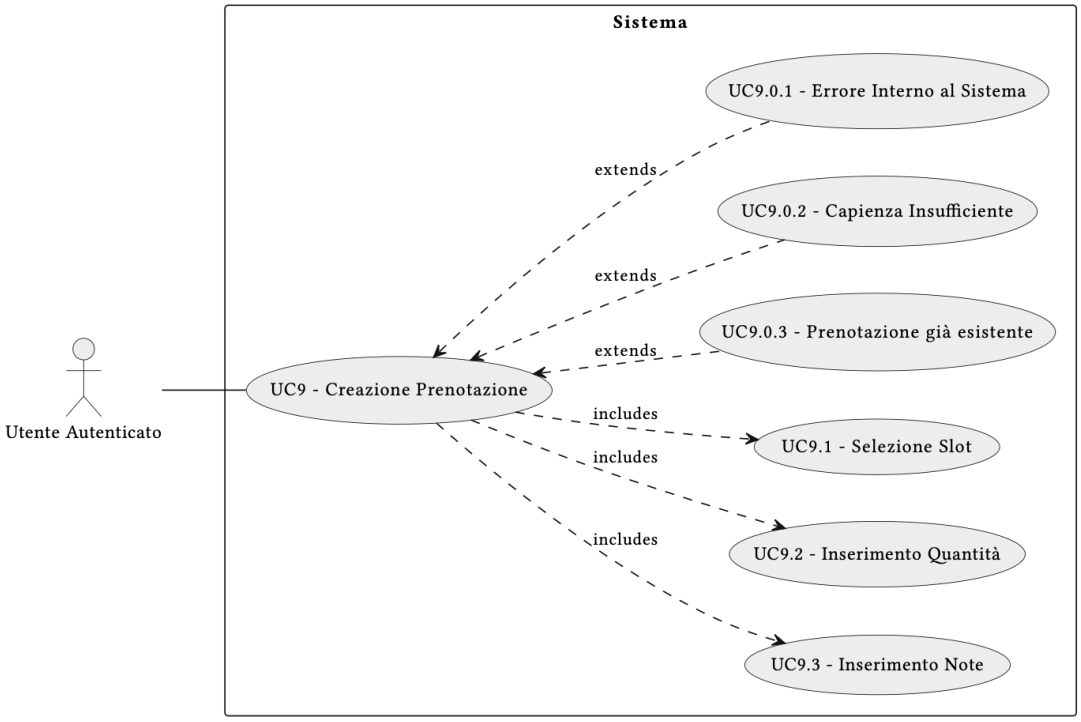

Figure 61: UC9 - Creazione Prenotazione

Figura 3.3: Estratto del diagramma dei casi d'uso generato tramite PlantUML.

Elaborazione originale dell'autore

Requisito	Descrizione	Caso d'Uso
RFOB59	Il Sistema deve eliminare in modo definitivo dal database tutti gli Slot che corrispondono ai filtri di ricerca forniti in input	[UC17]
RFOB60	Il Sistema deve applicare un'eliminazione a cascata, rimuovendo automaticamente e definitivamente tutte le prenotazioni associate agli Slot oggetto di eliminazione massiva	[UC17]
RFOB61	Il Sistema deve validare la coerenza temporale dei filtri, impedendo l'operazione se la Data di Fine precede la Data di Inizio	[UC17.3.1]
RFOB62	Il Sistema, qualora la Data di Inizio non venga fornita, deve utilizzare l'istante corrente come limite temporale inferiore di default	[UC17]
RFOB63	Il Sistema deve richiedere obbligatoriamente il parametro "Luogo di scarico" per consentire l'interrogazione degli Slot a database	[UC18.0.3]
RFOB64	Il Sistema deve applicare filtri di ricerca aggiuntivi (Tipo di Carico, Finestra Temporale) se valorizzati dall'utente	[UC18]

Figura 3.4: Estratto della tabella di tracciamento dei requisiti funzionali.

Elaborazione originale dell'autore

Inizialmente, la mia propensione per la documentazione formale e la totale assenza di conoscenze pregresse sui linguaggi C# e React hanno suscitato alcune perplessità nel *tutor* aziendale. Tuttavia, dopo aver esaminato la precisione dei documenti che ho prodotto, egli ha compreso il valore del rigore analitico e mi ha incitato a proseguire con questa metodologia.

Forte dei requisiti validati, in circa due settimane ho completato lo sviluppo di un primo prototipo funzionante esponendo alcuni degli *endpoint* fondamentali, con lo scopo di acquisire dimestichezza con lo *stack* tecnologico. A seguito di questa validazione iniziale, ho avviato la strutturazione ingegneristica dell'applicativo: la progettazione di dettaglio e la relativa codifica definitiva hanno richiesto approssimativamente un mese di lavoro aggiuntivo. Durante la stesura

dell'eseguibile preliminare, ho ideato il nome definitivo dell'applicativo — Kargo — e ne ho disegnato il logo ufficiale, che richiama la veste cromatica dell'azienda committente, come illustra la Figura 3.5.

Figura 3.5: Logo dell'applicativo Kargo.

Elaborazione originale dell'autore

3.3 Evoluzione del dominio applicativo

La rapidità con cui ho consegnato il prototipo ha fornito alla dirigenza tecnica l'opportunità di espandere il raggio d'azione di Kargo. Il *tutor* ha spostato l'attenzione dal semplice utilizzatore finale alla figura dell'amministratore, con l'intento di rendere il sistema autosufficiente e capace di autogestirsi integralmente.

Ho implementato funzionalità che garantiscono agli amministratori il controllo completo sulle prenotazioni e sugli intervalli temporali, e permettono loro di operare sia su singole istanze sia tramite elaborazioni massive. Nello specifico, ho sviluppato un algoritmo in grado di generare e istanziare in tempi rapidissimi almeno 500 *slot* temporali differenti per un singolo luogo di scarico, le cui prestazioni rimangono a discrezione del calcolo parametrico in funzione della mole dei dati richiesti. Per potenziare il supporto decisionale, ho integrato procedure di esportazione dei dati in formato `.pdf` e `.csv`, che consentono agli amministratori di estrarre *report* dettagliati delle prenotazioni associate a ogni specifico *slot*. Inoltre, per agevolare l'auto-formazione degli utenti, ho in-

trodotto un modulo di consultazione documentale che espone dinamicamente il manuale utente o il manuale amministratore in base al profilo di autorizzazione del soggetto autenticato.

Parallelamente, ho rilevato la necessità di innalzare gli standard di sicurezza. Ho integrato un sistema di validazione visiva tramite *Completely Automated Public Turing test to tell Computers and Humans Apart (CAPTCHA)*$_G$ — un *test* di sfida-risposta che i sistemi impiegano per determinare se l'utente sia un essere umano o un calcolatore — per scongiurare registrazioni automatizzate da parte di *bot*. Inoltre, il *tutor* ha imposto un meccanismo di attivazione degli *account* tramite *One Time Password (OTP)*$_G$ — codici numerici temporanei validi per una singola sessione di accesso — per prevenire furti d'identità aziendale. Invece di adottare un approccio di codifica standard, ho implementato l'algoritmo *BCrypt*$_G$ — una funzione crittografica di *hashing* che integra un meccanismo di rallentamento intenzionale per resistere agli attacchi di forza bruta — e ho configurato un *Work Factor* asimmetrico, esito di un preciso compromesso progettuale: ho applicato un costo pari a 13 per l'*hashing* pesante delle *password*, in modo da massimizzare la sicurezza a riposo, e un costo ridotto a 11 per i codici OTP, per ottimizzare la velocità di validazione in una finestra temporale ristretta a 15 minuti.

Per la gestione delle sessioni, ho contestato l'approccio iniziale che prevedeva il passaggio di dati sensibili in chiaro tramite richieste HTTP GET, e ho proposto e implementato in sua sostituzione l'adozione dei *JSON Web Token (JWT)*$_G$ — *token* crittografici compatti che garantiscono il trasferimento sicuro di informazioni tra le parti. La struttura scelta prevede un *Access Token* a vita breve (impostato a 15 minuti) e un *Refresh Token* a lunga durata (impostato a 7 giorni e rinnovato a ogni aggiornamento di sessione). Il sistema archivia entrambi i *token* all'interno di *cookie* protetti dal *flag HttpOnly*, una configurazione che li rende del tutto inaccessibili agli *script* in esecuzione lato *client*. Ho calcolato questa specifica taratura temporale per bilanciare sicurezza e *User Experience (UX)*$_G$ — l'insieme delle percezioni e delle risposte emotive che le persone sviluppano durante l'interazione con un sistema —: la finestra di 15 minuti minimizza l'esposizione del sistema in caso di furto del *token* (ad esempio

tramite attacchi di *Cross-Site Scripting (XSS)*$_G$ — vulnerabilità informatiche che consentono ai malintenzionati di iniettare codice malevolo all'interno di pagine *web* legittime —), mentre la persistenza di 7 giorni del *Refresh Token* evita ai corrieri la frustrazione di doversi riautenticare continuamente durante i turni lavorativi.

3.4 Progettazione architetturale e codifica

L'evoluzione dei requisiti ha progressivamente concentrato la complessità del sistema nelle regole di dominio, dinamica che ha reso inefficace un approccio procedurale classico. Ho pertanto adottato il modello di sviluppo guidato dal dominio (*Domain-Driven Design (DDD)*$_G$) — un paradigma architetturale che modella il *software* attorno alle complesse regole di *business* e promuove un linguaggio rigoroso e condiviso tra tecnici ed esperti di settore —, necessario per governare vincoli stringenti: le prenotazioni dovevano rispettare un tempo limite di registrazione; il sistema doveva condizionare dinamicamente le operazioni permesse in base allo stato dell'utente; la generazione *batch* richiedeva controlli incrociati per impedire sovrapposizioni e rispettare il limite massimo di porte disponibili.

Per proteggere questa logica di *business*, ho adottato l'architettura esagonale (*Ports and Adapters*) e ho enucleato cinque entità di dominio principali (`User`, `Booking`, `Slot`, `Place`, `LoadType`) che ho rigorosamente incapsulato e protetto tramite quattordici *Value Object* (quali `DoorCapacity`, `TimeWindow`, `Company` e `Quantity`) per garantire l'invarianza assoluta dei dati. L'utilizzo di interfacce — le cosiddette porte — ha delegato agli specifici *adapter* la responsabilità di dialogare con la base dati corretta, una scelta che rende il sistema agnostico rispetto alle differenze strutturali tra il *database* di collaudo e quello di produzione. Tramite PlantUML ho strutturato i diagrammi delle classi a partire dal *Domain Layer* fino ai *Controller* (Figura 3.6).

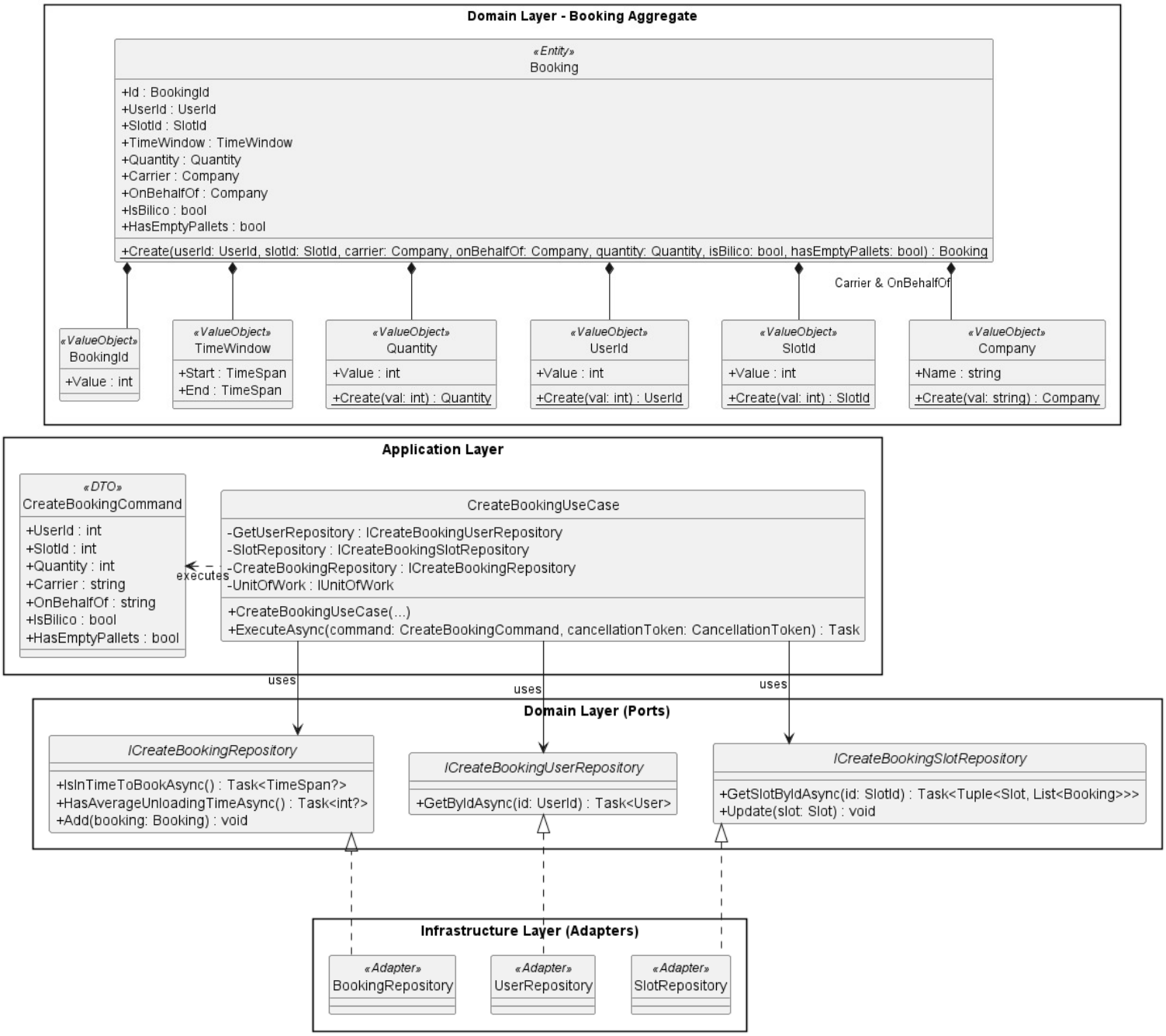

Figura 3.6: Modellazione delle classi di dominio e del livello applicativo per la creazione prenotazioni.

Elaborazione originale dell'autore

Nella stesura del codice, ho applicato i principi SOLID per la creazione di *software* manutenibile e disaccoppiato:

- **Inversione del controllo e iniezione delle dipendenze:** ho svincolato le classi dalla loro implementazione concreta mediante interfacce astratte,

per garantire modularità strutturale.

- **Segregazione delle Interfacce:** ho frammentato i contratti di accesso ai dati creando interfacce chirurgiche per ogni singolo caso d'uso (es. `IRegisterUserRepository`), per evitare classi monolitiche.
- **Responsabilità singola e Sostituzione di Liskov:** il codice di dominio interagisce con interfacce minimali e delega agli *adapter* la traduzione dei contratti astratti.

A supporto dei principi SOLID, ho consolidato l'architettura applicando in modo sistematico i *Design Pattern* dell'ingegneria del *software*. Nel *backend* in C#, il paradigma esagonale si fonda nativamente sul *pattern* ***Adapter***, che ho impiegato per disaccoppiare in modo netto la logica di dominio dalle infrastrutture esterna e di persistenza. Per garantire l'invarianza dei dati fin dall'istanziazione, ho utilizzato il ***Factory Method*** nella creazione delle Entità e dei *Value Object*, per centralizzare le logiche di validazione e impedire la generazione di stati inconsistenti. Ho modellato il flusso di esecuzione delle chiamate separando le operazioni di lettura da quelle di scrittura, in osservanza del principio ***Command and Query***. In questo contesto, i singoli *Use Case* agiscono come un ***Facade*** architetturale: essi costituiscono l'unico punto di accesso invocato dai *Controller*, per nascondere l'intera complessità dell'orchestrazione di dominio. Il *pattern Dependency Injection* governa interamente tale struttura e gestisce la risoluzione e il ciclo di vita dei servizi in base alle interfacce. Sul versante *frontend*, ho ingegnerizzato l'applicativo secondo il paradigma ***Component-Based*** e ho frammentato le interfacce utente in elementi atomici, riutilizzabili e indipendenti; ho così incapsulato la logica di presentazione per garantire una manutenzione modulare e scalabile.

Sul livello *frontend* in React e TypeScript, per la gestione delle comunicazioni di rete, ho costruito un *client* HTTP personalizzato che si basa su Axios, dal quale derivano specifici servizi *frontend* che mappano specularmente i *Controller* del *backend*. La definizione rigorosa di *Data Transfer Object (DTO)*$_G$ — oggetti strutturati privi di logica comportamentale che i sistemi impiegano

esclusivamente per incapsulare e trasportare informazioni tra i diversi livelli dell'applicazione — sia per le risposte che per le richieste garantisce l'integrità dei dati in transito. Per l'orchestrazione dello stato ho impiegato gli *hook* nativi di React (come `useState`), affiancati dal *custom hook* `useAuth` per la gestione della sessione utente. Per l'acquisizione e la validazione dei dati utente, ho integrato la libreria `react-hook-form`, che si avvale di Zod per la validazione schematica dei campi, per garantire robustezza direttamente lato *client*. Ho ingegnerizzato l'interfaccia utente massimizzando il riuso del codice: ho isolato componenti strutturali atomici (campi *input*, pulsanti e un *form* universale) che ho impiegato trasversalmente in tutte le nove pagine dell'applicativo (dall'autenticazione alla gestione degli *slot*). Ho ottenuto l'incapsulamento stilistico tramite l'adozione dei *CSS Modules*, strutturati per dominio di interesse (ad esempio `auth.module.css` e `bookings.module.css`), per prevenire collisioni globali e favorire la manutenibilità.

Infine, per preservare le prestazioni del *backend*, ho centralizzato la gestione degli errori tramite un `GlobalExceptionFilterAttribute` — evitando la duplicazione di costrutti `try-catch` in violazione del principio *Don't Repeat Yourself (DRY)*$_G$ — e ho implementato un *Rate Limiting*$_G$ personalizzato sulle API. Tale filtro analizza l'intestazione `X-Forwarded-For` per intercettare gli IP reali dietro eventuali *proxy* e ne salva lo stato nella memoria RAM. Ho parametrato le soglie in base alle metriche che ho stimato congiuntamente al *tutor*: un limite restrittivo di tre richieste ogni 60 secondi per le operazioni critiche di autenticazione, e una soglia permissiva di cinque operazioni ogni 120 secondi per la manipolazione massiva degli *slot*.

A titolo esemplificativo, la Figura 3.7 riporta l'implementazione del caso d'uso primario per la generazione delle prenotazioni. Il frammento evidenzia l'applicazione pratica dell'Iniezione delle Dipendenze — il sistema passa le interfacce di accesso ai dati direttamente al costruttore — e l'incapsulamento delle regole di *business*. Il sistema delega la persistenza agli *adapter* infrastrutturali tramite il *pattern Unit of Work* — che raggruppa più operazioni sui dati in un'unica transazione atomica — esclusivamente dopo aver validato i vincoli di dominio (quali lo stato dell'utente, la capienza dello *slot* e le tempistiche di

scarico).

```
public class CreateBookingUseCase
{
    2 riferimenti
    private readonly ICreateBookingUserRepository GetUserRepository;
    3 riferimenti
    private readonly ICreateBookingSlotRepository SlotRepository;
    4 riferimenti
    private readonly ICreateBookingRepository CreateBookingRepository;
    2 riferimenti
    private readonly IUnitOfWork UnitOfWork;

    0 riferimenti
    public CreateBookingUseCase(
        ICreateBookingUserRepository createBookingUserRepository,
        ICreateBookingSlotRepository createBookingSlotRepository,
        ICreateBookingRepository createBookingRepository,
        IUnitOfWork unitOfWork)
    {
        GetUserRepository = createBookingUserRepository;
        SlotRepository = createBookingSlotRepository;
        CreateBookingRepository = createBookingRepository;
        UnitOfWork = unitOfWork;
    }

    0 riferimenti
    public async Task ExecuteAsync(CreateBookingCommand command, CancellationToken cancellationToken = default)
    {
        UserId userId = UserId.Create(command.UserId);
        SlotId slotId = SlotId.Create(command.SlotId);
        Quantity quantity = Quantity.Create(command.Quantity);

        User user = await GetUserRepository.GetByIdAsync(userId, cancellationToken);
        if (user == null) throw new BusinessException("USER_NOT_FOUND", "User not Found.");
        if (user.UserStatus == Domain.Users.Enums.UserStatus.Blocked) throw new BusinessException("USER_BLOCKED", "This user is currently blocked.");

        (Slot slot, IEnumerable<Booking> bookings) = await SlotRepository.GetSlotByIdAsync(slotId, cancellationToken);
        if (slot == null) throw new BusinessException("SLOT_NOT_FOUND", "Slot not Found.");
        if (bookings.Any(b => b.UserId.Value == userId.Value)) throw new BusinessException("BOOKING_DUPLICATED", "Booking already exists.");
        if (slot.Capacity.AvailableDoors <= 0) throw new BusinessException("SLOT_FULL", "Maximum capacity reached for this slot.");

        TimeSpan? bookingDeadline = await CreateBookingRepository.IsInTimeToBookAsync();
        slot.ValidateBookingDeadline(DateTime.Now, bookingDeadline);

        int? averageUnloadingTime = await CreateBookingRepository.HasAverageUnloadingTimeAsync();
        slot.ValidateLoadTime(quantity, averageUnloadingTime);

        slot.Capacity.Decrease();
        SlotRepository.Update(slot);

        Booking booking = Booking.Create(userId, slotId, Company.Create(command.Carrier), Company.Create(command.OnBehalfOf), quantity, command.IsBilico, command.HasEmptyPallets);

        CreateBookingRepository.Add(booking);

        if (!await UnitOfWork.CommitAsync(cancellationToken)) throw new BusinessException("INTERNAL_SERVER_ERROR", "Errore durante l'aggiornamento dello slot.");
    }
}
```

Figura 3.7: Implementazione del caso d'uso di creazione prenotazione.

Elaborazione originale dell'autore

3.5 Verifica e validazione delle funzionalità

Ho interpretato l'attività di verifica come un elemento strutturale continuo del ciclo di sviluppo. Ho condotto la validazione dei requisiti in modo strettamente incrementale, e ho sottoposto ogni nuova funzionalità a riscontri operativi diretti per garantirne la piena conformità.

Per collaudare il *backend*, mi sono avvalso principalmente di *Postman* per simulare i percorsi che i diagrammi dei casi d'uso delineano; in questo modo ho costruito progressivamente una *collection* dedicata e organica che ricalca l'intero

spettro operativo del sistema: dalla registrazione e gestione dell'autenticazione, fino alle elaborazioni *Create, Read, Update, Delete (CRUD)*$_G$ — acronimo che identifica le quattro funzioni fondamentali di creazione, lettura, aggiornamento e cancellazione dei dati — singole e massive per *slot* e prenotazioni, con l'inclusione dei *test* di esportazione in formato `.pdf` e `.csv`. A questa validazione metodica ho affiancato Swagger per documentare e collaudare visivamente gli *endpoint* disponibili.

Sul versante *frontend*, ho validato l'applicativo tramite esecuzioni dirette nell'ambiente di sviluppo locale e mi sono focalizzato sull'accessibilità visiva e comportamentale con il supporto dei DevTools di Google Chrome. Ho verificato la reattività (*responsiveness*) dell'applicativo su schermi *desktop*, *tablet* e terminali mobili.

Sebbene abbia strutturato l'ambiente TypeScript per supportare la scrittura di *unit test* al fine di garantire una solida base per future integrazioni, la validazione dell'applicativo si è basata su un collaudo empirico iterativo; ho poi integrato tale verifica con le revisioni periodiche insieme al *tutor* aziendale per stabilire l'approvazione funzionale dei moduli. Questa decisione deriva da un preciso compromesso che ho stabilito con la dirigenza: data la forte spinta verso il rapido rilascio che il committente avanzava per validare l'idea, la direzione ha preferito privilegiare la consegna funzionale del prodotto rispetto all'onere temporale che la stesura di una *suite* completa di *test* automatizzati comporta, e ha rimandato quest'ultima alle future iterazioni di manutenzione.

3.6 Risultati qualitativi e quantitativi

Al termine delle ore previste per la stesura del codice, ho sottoposto l'applicativo Kargo alla valutazione finale della dirigenza tecnica. L'esito del progetto si articola su due dimensioni d'analisi: la percezione del prodotto dal lato utente e la misurazione delle metriche ingegneristiche di copertura.

3.6.1 Il prodotto dal lato utente

Sul piano qualitativo, Kargo si presenta come un applicativo reattivo, solido e orientato all'usabilità. Sebbene abbia sviluppato l'applicativo per l'uso interno logistico — un settore che spesso subordina l'estetica alla pura funzionalità — ho dedicato particolare attenzione all'interfaccia utente e ho garantito il rispetto degli standard di accessibilità *Web Content Accessibility Guidelines (WCAG)*$_G$ — linee guida internazionali che definiscono i criteri tecnici necessari per rendere i contenuti digitali fruibili in modo inclusivo, in particolare alle persone con disabilità — di livello AA e, in alcuni componenti precisi, di livello AAA. Ho verificato tale conformità attraverso l'analisi dei rapporti di contrasto cromatico, l'implementazione della navigazione completa tramite tastiera e l'impiego dei *tag* semantici HTML5, che ho esaminato mediante gli strumenti di ispezione del *browser*.

L'interfaccia guida gli attori del sistema attraverso flussi operativi chiari. Per gestire la notevole mole di dati che deriva dall'espansione del dominio, ho progettato e implementato funzioni avanzate di ricerca e filtraggio dinamico. Sia i corrieri, nella consultazione degli *slot* liberi, sia gli amministratori, nel monitoraggio globale del deposito, possono interrogare la base dati in tempo reale, isolando le prenotazioni per data, stato o tipologia di carico con risposte visive immediate.

La Figura 3.8 mostra il flusso lato corriere: la selezione di una finestra temporale libera tra quelle disponibili nel deposito, propedeutica alla creazione della prenotazione che la Figura 3.9 illustra, dove il corriere conferma i dati del proprio mezzo e del carico trasportato. Sul lato amministrativo, la Figura 3.10 illustra la vista di monitoraggio complessivo degli *slot* del deposito, mentre la Figura 3.11 mostra il pannello con cui l'amministratore modifica i vincoli di un singolo intervallo, quali la capienza e la tipologia di carico consentita.

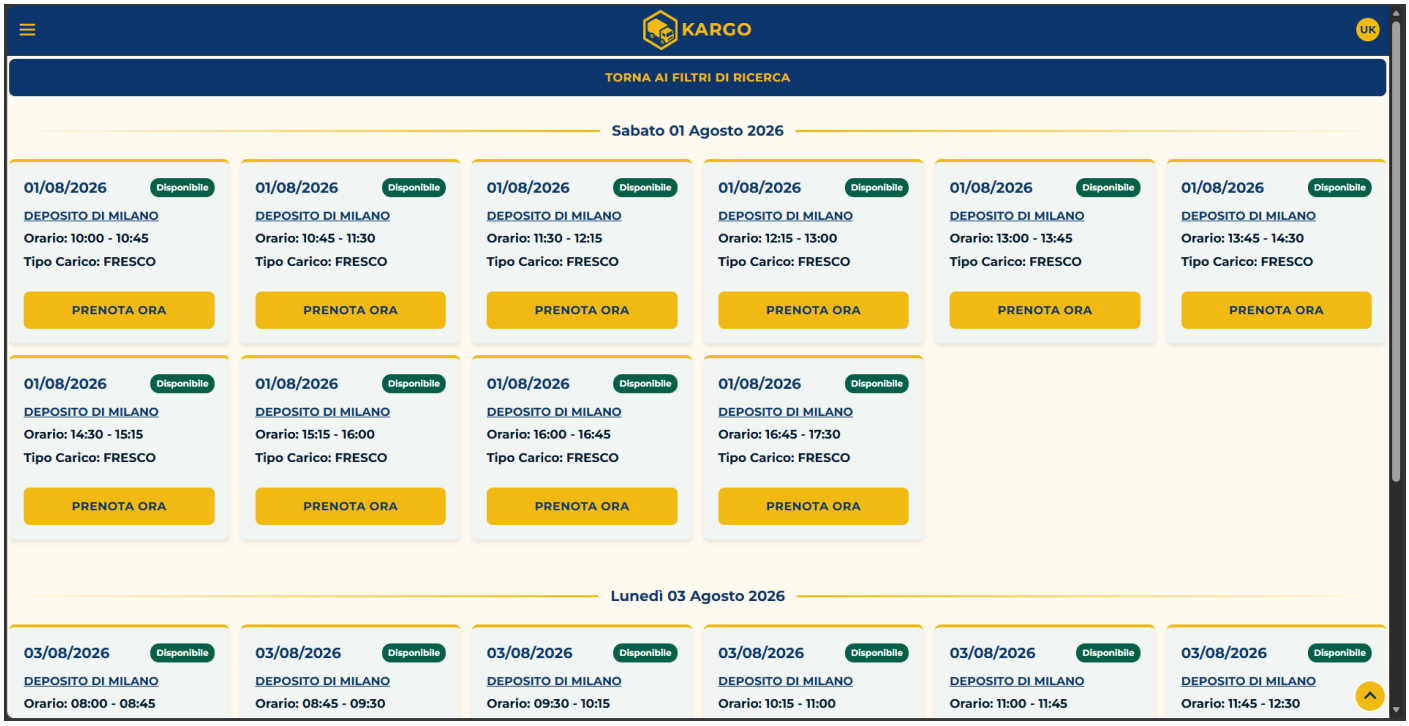

Figura 3.8: Interfaccia di selezione della finestra temporale di prenotazione.

Elaborazione originale dell'autore

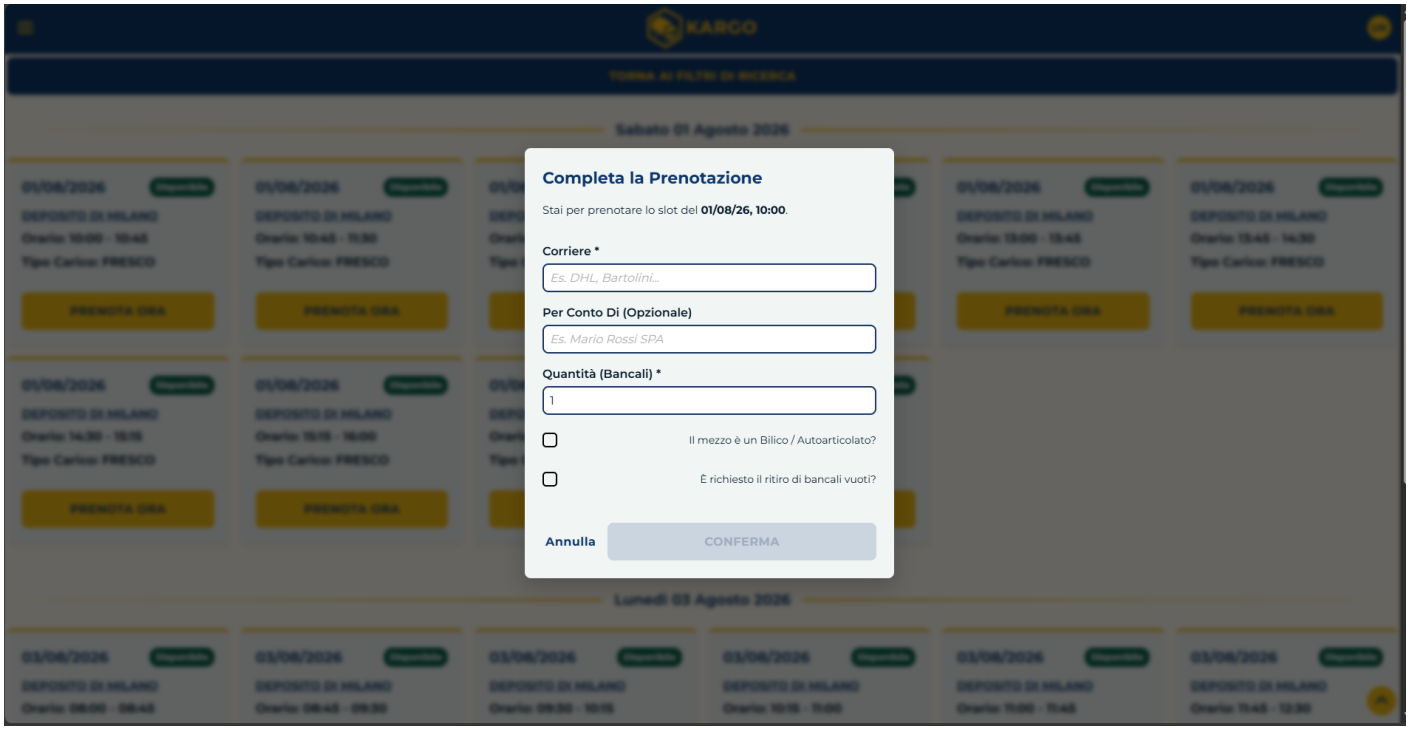

Figura 3.9: Dettaglio sul completamento della prenotazione.

Elaborazione originale dell'autore

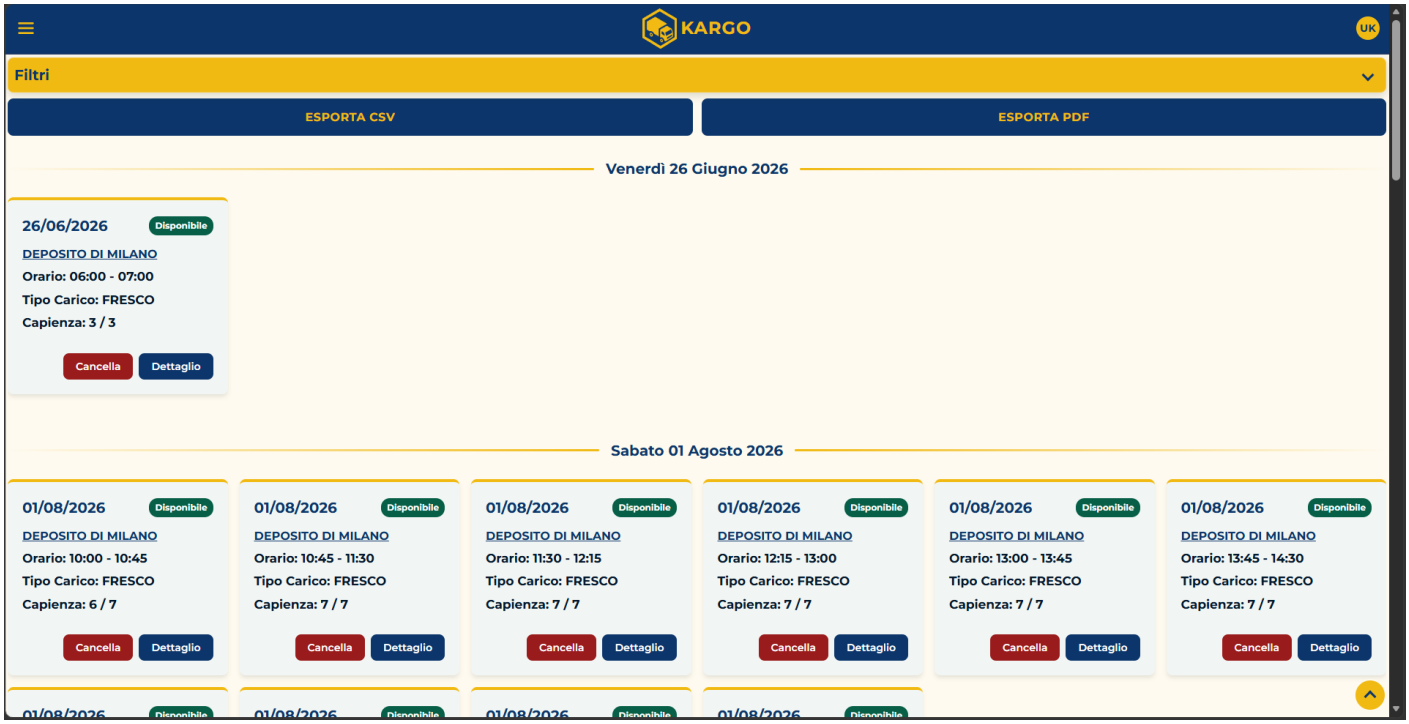

Figura 3.10: Visualizzazione della gestione degli slot temporali.

Elaborazione originale dell'autore

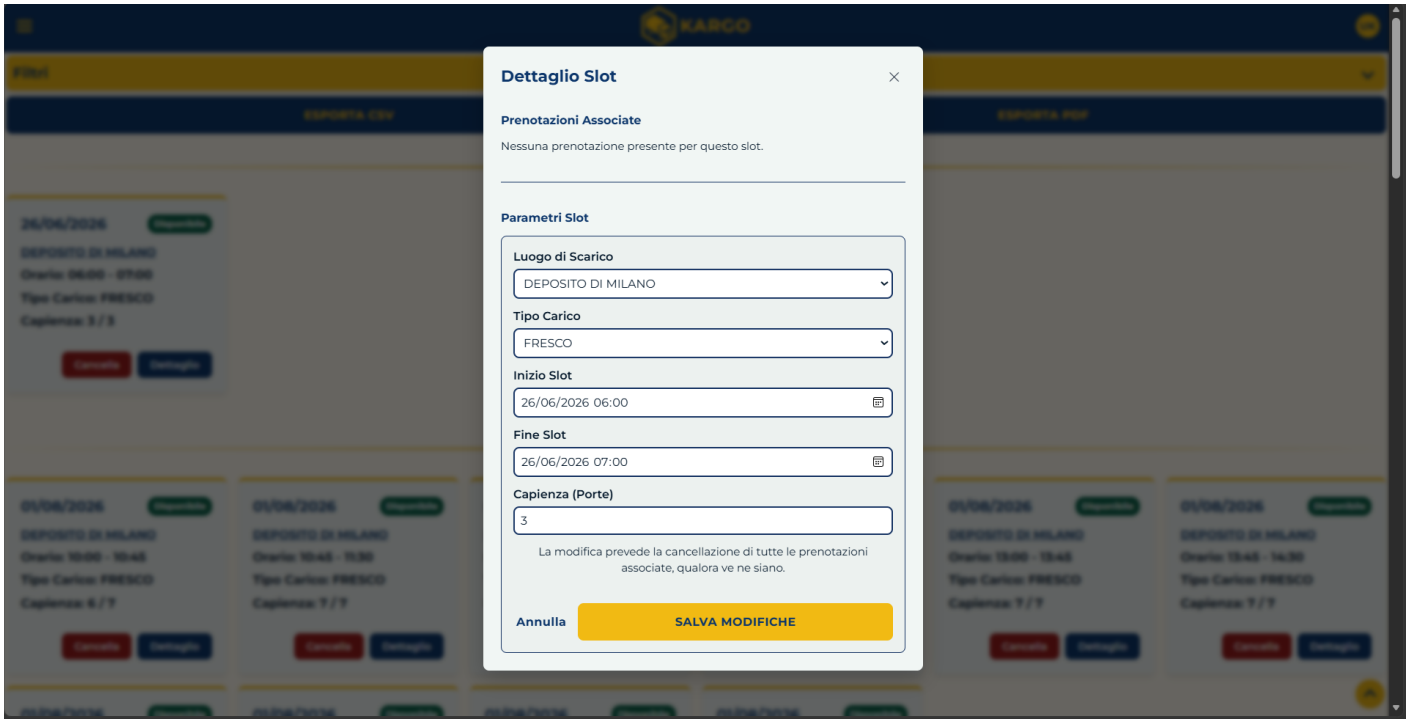

Figura 3.11: Dettaglio sulla gestione degli slot temporali.

Elaborazione originale dell'autore

3.6.2 Copertura e metriche di progetto

Sotto il profilo quantitativo, i risultati testimoniano il conseguimento degli obiettivi aziendali che il Capitolo 2 delinea. Nonostante il *tutor* abbia richiesto un ampliamento del dominio a lavori già avviati, ho consegnato la *release* definitiva dell'applicativo con circa una settimana di anticipo rispetto alle scadenze originarie che ho originariamente stabilito con l'azienda. Tale flessibilità operativa e la rapidità di adattamento alle nuove richieste non derivano dal caso, ma rappresentano il beneficio diretto dell'adozione dell'architettura esagonale e dei principi SOLID poiché il forte disaccoppiamento dei componenti mi ha permesso di integrare i nuovi requisiti amministrativi senza generare regressioni nel codice già consolidato.

Dal punto di vista della copertura funzionale, ho mappato e soddisfatto le richieste del committente. La Tabella 3.1 illustra il riepilogo quantitativo dei requisiti individuati in analisi, suddivisi per tipologia e grado di adempimento.

Tipologia Requisito	**Obbligatori**	**Desiderabili**	**Opzionali**
Requisiti Utente	24	1	1
Requisiti Funzionali	80	1	0
Requisiti non Funzionali	3	1	2
Requisiti di Interfaccia	35	3	1
Totale	**142**	**6**	**4**

Tabella 3.1: Riepilogo quantitativo della tracciabilità dei requisiti di progetto.

Valori assoluti derivati dalla matrice di tracciamento dei requisiti validata dal *tutor* aziendale.

A livello di volumetria del prodotto, il lavoro ingegneristico si concretizza in un artefatto *software* di oltre 8.000 righe di codice (LOC), che ho ripartito equamente tra le robuste architetture di *backend* (circa 4.500 LOC) e le dinamiche interfacce *frontend* (circa 3.500 LOC). Il *repository* finale si compone di esattamente 28 *endpoint* API lato *backend*, che ho suddiviso logicamente in sei *Controller RESTful*. Questa elevata granularità costituisce una scelta proget-

tuale intenzionale per rispettare i vincoli del modello *RESTful*: anziché creare pochi *endpoint* monolitici, ho preferito frammentare le risorse per ridurre il carico delle singole chiamate HTTP e per favorire il riuso delle logiche da parte del *frontend*. Pur in assenza di metriche automatizzate di *code coverage* a causa dei vincoli di *time-to-market*, ho garantito una copertura funzionale del 100% sulle API: le chiamate della *collection Postman* collaudano infatti esaustivamente ciascuno dei 28 *endpoint* che il *backend* espone.

L'infrastruttura si avvale inoltre di cinque entità di dominio in C# e di svariate decine di componenti *custom* in React, che ho orchestrato su nove viste applicative che ho sviluppato *ex novo*. A questa massiccia produzione informatica ho affiancato la redazione formale del documento di analisi dei requisiti iniziale e la stesura di due guide operative complete: il manuale utente per i corrieri e il manuale amministratore per il personale della logistica interna. La Tabella 3.2 dettaglia la tracciabilità e la metodologia empirica per ciascun nucleo del sistema; su queste basi, la validazione finale registra l'esito positivo dei collaudi empirici e visivi che ho condotto congiuntamente al *team* tecnico di Ergon Informatica, a conferma della piena aderenza del prototipo alle specifiche.

Tabella 3.2: Matrice di implementazione e collaudo atomico dei requisiti applicativi.

Requisito atomico	**Criterio di verifica oggettivo (Come)**	**Convalida empirica e tempistica (Da chi e Quando)**
Dominio Utente (perimetro PoC iniziale)		
Gestione Identità: Registrazione	Controllo della risposta `201` su `POST /api/auth/register` e *test* di innesco *e-mail* su 4 scenari alternativi.	Esecuzione autonoma dei *test* di carico su Postman con superamento di tutti i 4 scenari alla quinta settimana.
Gestione Identità: Attivazione	Verifica della validità del *token* con riscontro di restituzione `200` o codice di errore `410`.	Verifica empirica di tutte le risposte HTTP effettuata in autonomia dall'autore alla quinta settimana.

Continua nella pagina successiva...

Tabella 3.2 – Continua dalla pagina precedente

Requisito atomico	**Criterio di verifica oggettivo (Come)**	**Convalida empirica e tempistica (Da chi e Quando)**
Gestione Identità: Revoca	Ispezione su `DELETE /api/users/{id}` per l'invalidazione immediata della sessione attiva.	Ispezione diretta dello stato di sessione completata in autonomia alla sesta settimana.
Autenticazione (Login/Logout)	Validazione dell'emissione della stringa JWT (`200`) e del relativo blocco selettivo (`401`).	Esecuzione dei *test* di emissione e cancellazione JWT su Postman alla quinta settimana.
Sicurezza: Recupero password	Ispezione del blocco di sicurezza (`403`) e controllo della validità temporale del collegamento.	Verifica dei blocchi temporali e contestuale riscontro positivo del *tutor* alla sesta settimana.
Gestione Prenotazioni (API CRUD)	Ispezione dei codici di stato HTTP (`200`/`201`/`204`) e verifica della struttura del *payload* JSON.	Ispezione dei codici HTTP e riscontro della perfetta conformità dei contratti DTO alla sesta settimana.
Dominio Amministratore (espansione non preventivata)		
Modellazione Slot: Creazione singola	Analisi su `POST /api/slots` per verificare l'inserimento ed il rigetto automatico dei conflitti (`409`).	Analisi di persistenza con accertamento del blocco dei conflitti temporali alla settima settimana.
Modellazione Slot: Creazione *batch*	Misurazione della latenza del *server* per l'elaborazione massiva di un lotto di 100 unità di *slot*.	Misurazione dei tempi di risposta del *server* con riscontro di valori inferiori ai 2 secondi alla settima settimana.

Continua nella pagina successiva...

Tabella 3.2 – Continua dalla pagina precedente

Requisito atomico	**Criterio di verifica oggettivo (Come)**	**Convalida empirica e tempistica (Da chi e Quando)**
Propagazione Modifiche (Slot multipli)	Ispezione a *database* per accertare la propagazione delle modifiche alle prenotazioni collegate.	Verifica dell'integrità referenziale a *database* sul blocco delle modifiche a cascata alla settima settimana.
Gestione Conflitti: Eliminazione forzata	Verifica del blocco logico di conferma e innesco della notifica automatica verso il corriere.	Controllo del blocco logico e delle notifiche con visto finale del *tutor* all'ottava settimana.
Documentazione		
Manuale operativo Utente	Revisione incrociata dei flussi utente descritti rispetto alle funzionalità reali implementate nel sistema.	Revisione incrociata dei flussi utente con approvazione finale del *tutor* all'ottava settimana.
Manuale operativo Amministratore	Revisione incrociata dei flussi amministrativi descritti rispetto alle interfacce gestionali realizzate.	Revisione dei flussi gestionali con approvazione formale della dirigenza all'ottava settimana.

Capitolo 4

Valutazione retrospettiva

Nel presente capitolo conclusivo analizzo criticamente l'esperienza di stage. *Attraverso l'impiego di parametri oggettivi, verifico il grado di raggiungimento degli obiettivi aziendali e formativi di riferimento. Esamino poi la maturazione professionale che ho sviluppato sul campo, per tracciare infine un bilancio sulla distanza che separa la preparazione teorica offerta dal corso di studi in Informatica e le competenze ingegneristiche che il mercato del lavoro richiede.*

4.1 Soddisfacimento degli obiettivi

La valutazione del tirocinio si fonda sul grado di completamento degli scopi formativi e operativi che il progetto di *stage* definisce. Dopo l'analisi dei requisiti funzionali e del loro collaudo nel capitolo precedente, in questa sede misuro oggettivamente le competenze tecniche che ho maturato e i traguardi architetturali che ho conseguito.

Ho organizzato lo sviluppo in modo da coprire tutte le richieste aziendali e ho concluso l'intero ciclo di prototipazione con una settimana di anticipo rispetto alla scadenza curriculare. La Tabella 4.1 sintetizza questo bilancio: essa formalizza esplicitamente le metodologie di verifica, le tempistiche di convalida e i soggetti responsabili della valutazione tecnica, al fine di escludere valutazioni autoreferenziali o prive di un riscontro procedurale.

Tabella 4.1: Matrice di valutazione e validazione empirica degli obiettivi di *stage*.

Obiettivo di *stage*	**Metodologia ed evidenza di verifica (Come)**	**Soggetto valutatore e tempistica (Da chi e Quando)**
Sviluppo prototipo *Full-Stack* *(Aziendale)*	Ispezione visiva dei flussi d'interfaccia ed esecuzione dei *test* d'integrazione sulle API per i moduli Utente e Amministratore.	Validazione diretta del *tutor* aziendale e del *team* tecnico durante il collaudo finale (ottava settimana).
Padronanza dello *stack* tecnologico *(Formativo)*	Verifica della stesura autonoma della *codebase* C# e React, misurata tramite la progressiva integrazione di componenti complessi senza supervisione esterna.	Certificazione a cura del *tutor* aziendale mediante revisioni del codice a cadenza settimanale per l'intera durata dello *stage*.
Progettazione della persistenza dati *(Formativo)*	Analisi dello schema logico del *database* e verifica del rispetto dei vincoli ACID tramite l'ORM Entity Framework e il *pattern Unit of Work*.	Approvazione del referente tecnico aziendale in sede di revisione architetturale (quarta settimana).
Ottimizzazione delle tempistiche *(Aziendale)*	Confronto analitico tra il cronoprogramma iniziale e la data di consegna dell'eseguibile definitivo per la logistica.	Rilevazione formale del *tutor* aziendale alla conclusione del modulo amministrativo (settima settimana).
Gestione del ciclo di vita del *software* *(Formativo / Aziendale)*	Ispezione e collaudo pratico dei manuali operativi redatti, verificandone l'efficacia tramite *test* di utilizzo assistito del personale interno.	Sottoscrizione e approvazione da parte della dirigenza aziendale durante l'allineamento conclusivo dell'attività di tirocinio.

Ho superato la scarsa familiarità iniziale con i *framework* in esame: questo risultato costituisce l'indicatore più tangibile del mio avanzamento tecnico. La traduzione di requisiti astratti in un'architettura completa e funzionante mi

permette di certificare il conseguimento del principale obiettivo formativo alla base di questo percorso.

4.2 Maturazione professionale

Nel corso dell'inserimento nelle dinamiche produttive quotidiane, ho abbandonato un approccio accademico esplorativo in favore di un rigore ingegneristico operativo, che ho sviluppato lungo tre direttrici principali: l'ingegnerizzazione del dominio, la resilienza del codice e l'autonomia nel *problem solving*.

Sotto il profilo architetturale, ho superato il modello a classi anemiche — struttura che caratterizzava frequentemente le mie prime iterazioni didattiche — attraverso l'applicazione dei principi del *Domain-Driven Design*. Ho orientato l'evoluzione della modellazione verso entità di dominio ricche e autosufficienti, in cui gli oggetti `Slot` e `Booking` incapsulano interamente la complessa logica di *business* logistica. Nell'entità `Slot`, ad esempio, ho implementato la protezione attiva degli invarianti: la classe stessa valuta il rispetto delle finestre temporali di disdetta, calcola la capacità di scarico e aggiorna dinamicamente lo stato delle porte disponibili. Attraverso questa architettura ho garantito la coerenza del dato senza delegare i controlli logici ai *controller* esterni.

Parallelamente, nella stesura del codice ho abbandonato la ricerca del solo caso d'uso ideale (*happy path*) e ho adottato i costrutti della programmazione difensiva. Ho tradotto l'anticipazione delle anomalie in un duplice livello di protezione: ho imposto una validazione rigorosa dei DTO in ingresso e ho isolato al contempo il *database* — che ho trattato sistematicamente come un servizio esterno potenzialmente instabile — per intercettare in sicurezza ogni eccezione di persistenza.

Ho acquisito una solida autonomia decisionale, che si traduce nella capacità di individuare vulnerabilità architetturali e di proporre soluzioni ingegneristiche proattive. L'implementazione di un'autenticazione *stateless* tramite *token* JWT, la configurazione di un sistema di *rate limiting*, la protezione degli *endpoint* secondo i ruoli e l'introduzione di un *Global Exception Handler* traggono origine

da un'analisi autonoma del sistema, che ho strutturato come *proof of concept* e che la dirigenza ha successivamente approvato.

Infine, sul piano metodologico, ho corretto la tendenza iniziale a formulare obiettivi giornalieri eccessivamente ampi e ho adottato la logica del metodo *C-V-R* (Completato, Verificato, Revisionato). Invece di verificare semplicemente il generico funzionamento di un componente, ho integrato verifiche oggettive sulle metriche prestazionali, sulla conformità strutturale dei contratti in transito sulla rete e sulla resilienza del sistema tramite l'iniezione intenzionale di dati non conformi. Questo cambio di paradigma mi ha permesso di trasformare gli ostacoli tecnici in problemi isolabili, una scelta che ha aumentato la velocità di sviluppo e l'affidabilità del prodotto finale.

4.3 Fondamenti teorici e applicazioni pratiche

Il confronto tra l'ambiente accademico e quello produttivo ha evidenziato quali nozioni del mio percorso di laurea trovassero un'applicazione concreta, e ha permesso di misurare la distanza tra la preparazione universitaria e gli standard di mercato odierni.

Durante l'esecuzione del progetto ho impiegato diverse competenze teoriche assimilate in sede universitaria, le quali mi hanno permesso di velocizzare la risoluzione di complessi problemi architetturali e implementativi:

- **Ingegneria del *Software*:** ho utilizzato i principi di *system design* appresi per governare l'intero ciclo di vita del prodotto e ho applicato in modo sistematico le direttive SOLID e i principali *design pattern* per ottenere un elevato disaccoppiamento dei moduli.

- **Basi di Dati:** ho adottato la teoria dei modelli relazionali per progettare lo schema logico e, grazie alla padronanza delle proprietà ACID, ho configurato l'ORM Entity Framework e ho implementato in sicurezza il *pattern Unit of Work*.

- **Tecnologie *web* e Programmazione ad oggetti:** le nozioni di base su HTML, CSS e JavaScript hanno abbattuto la mia curva di apprendimento

sull'ecosistema React, mentre le fondamenta di programmazione a oggetti mi hanno permesso di padroneggiare rapidamente la sintassi tipizzata del linguaggio C#.

- **Metodi e tecnologie per lo sviluppo *software*:** ho applicato l'approccio metodologico del corso e ho documentato le interfacce API, ho tracciato i casi limite di fallimento e ho predisposto l'applicativo per future integrazioni in *pipeline* di *testing* automatizzato.

4.3.1 Lacune formative e prospettive

La sfida principale all'avvio dello *stage* non ha riguardato l'apprendimento di nuovi linguaggi — dinamica fisiologica nella nostra professione — bensì l'integrazione di ecosistemi tecnologici disaccoppiati in un'unica architettura. Ho dovuto acquisire una trasversalità tecnica che i programmi universitari affrontano spesso in forma isolata o parziale.

Nel dettaglio, mi sono formato in autonomia su ambiti nevralgici: la costruzione di API REST, l'autenticazione tramite *token* JWT, il governo delle *policy* *Cross-Origin Resource Sharing (CORS)*$_G$ — meccanismo di sicurezza basato su intestazioni HTTP che permette ai *server* di specificare quali domini esterni siano autorizzati ad accedere alle proprie risorse —, l'allineamento dei contratti DTO tra *client* e *server*, e l'impiego di strumenti professionali come Postman e Swagger. Queste specifiche aree di integrazione, seppur fondamentali, trovano uno spazio applicativo ancora limitato nel piano didattico.

Sono consapevole della difficoltà di condensare la vasta panoramica dell'ecosistema IT nei limiti temporali di una laurea triennale. Ritengo tuttavia che l'introduzione di laboratori focalizzati sulla costruzione di architetture *web full-stack* fornirebbe agli studenti in Informatica le chiavi operative per decodificare più rapidamente le interazioni di rete tipiche dei sistemi aziendali.

4.4 Conclusioni

Attraverso il tirocinio curriculare presso Ergon Informatica Srl ho vissuto il culmine pratico dell'intero percorso formativo. Tramite la progettazione da zero dell'applicativo Kargo, ho interagito direttamente con l'intero ciclo di vita di un prodotto *software*, e ho gestito e risolto problemi strutturali e operativi all'interno di un contesto produttivo reale.

Dopo la misurazione del raggiungimento degli obiettivi aziendali che ho definito a inizio *stage*, confermo il successo del progetto, culminato nella consegna dell'artefatto funzionale con una settimana di anticipo rispetto al piano prestabilito. Parallelamente, avendo governato tutte le interfacce *frontend* e *backend*, ho raggiunto anche i miei obiettivi formativi personali: ho colmato il disorientamento tecnologico iniziale e ho acquisito piena autonomia di sviluppo.

Grazie a questa esperienza ho compreso l'importanza delle fondamenta teoriche erogate dal Dipartimento e ho individuato quali componenti di integrazione di rete ho dovuto affinare sul campo. Ritengo che il tirocinio abbia arricchito il mio bagaglio accademico con un metodo di lavoro empirico e verificabile, e mi abbia permesso di affrontare con maggiore maturità ingegneristica il futuro ingresso nel settore IT.

Acronimi e abbreviazioni

API Application Programming Interface. 15

B2B Business-to-Business. 14

CAPTCHA Completely Automated Public Turing test to tell Computers and Humans Apart. 30

CI/CD Continuous Integration / Continuous Delivery. 8

CORS Cross-Origin Resource Sharing. 48

CRM Customer Relationship Management. 4

CRUD Create, Read, Update, Delete. 36

DDD Domain-Driven Design. 31

DRY Don't Repeat Yourself. 34

DTO Data Transfer Object. 33

ERP Enterprise Resource Planning. 2

HTTP Hypertext Transfer Protocol. 23

IT Information Technology. 1

JWT JSON Web Token. 30

LLM Large Language Model. 10

MVP Minimum Viable Product. 18

OCX OLE Control Extension. 2

ORM Object-Relational Mapping. 14

OTP One Time Password. 30

PoC Proof of Concept. 11

R&D Research and Development. 11

RDBMS Relational Database Management System. 2

RESTful Representational State Transfer. 14

TCP/IP Transmission Control Protocol/Internet Protocol. 2

UML Unified Modeling Language. 23

UX User Experience. 30

VCS Version Control System. 10

WCAG Web Content Accessibility Guidelines. 37

XSS Cross-Site Scripting. 31

Glossario

Application Programming Interface (API) Contratto formale che definisce le modalità di comunicazione tra componenti *software* distinti. Le API moderne superano la pura funzione tecnica: costituiscono prodotti a sé stanti che espongono i servizi aziendali, favoriscono l'integrazione con sistemi di terze parti e alimentano l'economia dei microservizi.. i, 15

Batch Paradigma di elaborazione asincrona in cui il sistema esegue blocchi sequenziali di operazioni senza richiedere l'interazione umana. Risulta imprescindibile per le elaborazioni ad alta intensità computazionale, come l'addestramento di modelli di *machine learning*, le riconciliazioni bancarie notturne e la migrazione massiva di dati.. 2

BCrypt Funzione crittografica di *hashing* specializzata nell'archiviazione sicura delle *password*. Deriva dal cifrario Blowfish e implementa un rimescolamento dinamico (*salt*) unito a un fattore di costo (*work factor*) adattivo. Tale configurazione contrasta le capacità di calcolo crescente delle moderne schede grafiche e frustra gli attacchi di tipo *rainbow table*.. 30

Business-to-Business (B2B) Modello di transazione digitale tra imprese, caratterizzato da logiche di scambio diametralmente opposte a quelle per il consumatore finale. Tali architetture richiedono un'elevata automazione procedurale, l'esposizione di API di integrazione e l'adesione a severi standard di autenticazione federata.. i, 14

Client-Server Paradigma informatico che divide i carichi di elaborazione. Il *server* centralizza la logica di *business* e la protezione dei dati, mentre i

client periferici gestiscono la presentazione visiva e l'interazione con l'utente. Questa separazione strutturale facilita la manutenzione e la scalabilità orizzontale dei servizi.. 2

Code Coverage Metrica quantitativa che esprime la percentuale di codice sorgente effettivamente attraversata dagli *unit test* automatizzati. Benché l'industria la consideri uno standard di qualità, una copertura del cento per cento non rileva difetti logici o di progettazione, motivo per cui i *team* esperti vi affiancano verifiche architetturali e collaudi di integrazione.. 8

Completely Automated Public Turing test to tell Computers and Humans Apart (
Misura di sicurezza crittografica e comportamentale che i portali *web* impiegano per discriminare le interazioni umane dal traffico generato da *bot*. I modelli più recenti analizzano schemi di navigazione invisibili per ridurre l'attrito durante l'autenticazione.. i, 30

Continuous Integration / Continuous Delivery (CI/CD) Dorsale automatizzata della cultura DevOps che orchestra l'integrazione del codice. L'esecuzione sistematica di analisi statiche, *test* unitari e procedure di rilascio a ogni *commit* abbatte il rischio di regressioni e garantisce un codice costantemente pronto per la produzione.. i, 8

Create, Read, Update, Delete (CRUD) Costituisce il paradigma fondazionale per la gestione della persistenza dei dati. Nelle architetture *RESTful*, gli sviluppatori mappano tipicamente queste quattro operazioni sui verbi semantici del protocollo HTTP (POST, GET, PUT/PATCH, DELETE) per garantire interfacce uniformi e prevedibili.. i, 36

Cross-Origin Resource Sharing (CORS) Meccanismo di sicurezza basato su intestazioni HTTP che permette ai *server* di allentare le restrizioni della *Same-Origin Policy* (SOP). I *browser* lo impiegano per autorizzare e gestire in sicurezza le richieste trans-origine, bloccando le chiamate verso domini non esplicitamente consentiti dall'infrastruttura di destinazione.. i, 48

Cross-Site Scripting (XSS) Vulnerabilità di sicurezza tipica delle applicazioni *web* che affligge il livello di presentazione. I malintenzionati sfruttano la mancata sanificazione degli *input* per iniettare *script* esecutivi nel *browser* delle vittime, un'azione che compromette le sessioni attive e facilita l'esfiltrazione di *token* di accesso.. ii, 31

Data Transfer Object (DTO) Modello architetturale che ottimizza le comunicazioni di rete. Un DTO funge da contenitore serializzabile, del tutto privo di logica di *business*, che aggrega dati provenienti da entità molteplici. Il suo impiego riduce il numero di chiamate remote e nasconde la struttura interna del *database* ai *client* esterni.. i, 33

Domain-Driven Design (DDD) Paradigma ingegneristico che affronta la complessità del *software* ancorando il codice al dominio applicativo. Introdotto da Eric Evans, impone un Linguaggio Ubiquo condiviso tra sviluppatori ed esperti di settore. La separazione in *Bounded Contexts* isola le logiche di *business* e previene l'entropia architetturale nei sistemi *enterprise* distribuiti.. i, 31

Don't Repeat Yourself (DRY) Principio di progettazione che vieta la duplicazione delle logiche di *business* e dei dati. L'architettura deve prevedere una singola fonte di verità (*Single Source of Truth*) per ogni conoscenza di sistema, una pratica che abbatte i costi di manutenzione e previene le incongruenze durante gli aggiornamenti.. i, 34

Enterprise Resource Planning (ERP) Piattaforma centralizzata che governa i flussi logistici, finanziari e produttivi di un'azienda. Gli ERP contemporanei superano la pura archiviazione contabile: si affidano ad architetture *cloud* per offrire moduli analitici intelligenti, interoperabilità avanzata e visibilità in tempo reale sui processi industriali.. i, 2

Firewall Sistema di sicurezza perimetrale che ispeziona il traffico di rete in transito. Le implementazioni moderne di *Next-Generation Firewall* (NGFW) superano il semplice filtro di porte e protocolli: analizzano pro-

fondamente il carico utile delle applicazioni e integrano meccanismi di prevenzione delle intrusioni per neutralizzare le minacce avanzate.. 4

Geomarketing Disciplina analitica che fonde le scienze commerciali con l'intelligenza spaziale. L'elaborazione in tempo reale di flussi geospaziali permette alle aziende di mappare i bacini di utenza, ottimizzare la logistica delle consegne e indirizzare campagne di acquisizione con estrema precisione territoriale.. 4

Hypertext Transfer Protocol (HTTP) Protocollo applicativo *stateless* che regola la trasmissione dei documenti ipertestuali sulla rete Internet. L'evoluzione verso gli standard HTTP/2 e HTTP/3 ha introdotto il parallelismo delle richieste e la compressione delle intestazioni per abbattere la latenza nelle architetture moderne.. i, 23

Information Technology (IT) Macro-settore ingegneristico che progetta e gestisce l'infrastruttura tecnologica globale. Oltre al tradizionale supporto operativo, oggi l'IT ingloba la modernizzazione dei processi aziendali, l'ingegneria del *software* distribuito e la protezione degli *asset* informatici attraverso pratiche di *cybersecurity*.. i, 1

JSON Web Token (JWT) Standard aperto per la creazione di *token* di accesso che asseriscono dichiarazioni in formato JSON. La firma crittografica garantisce l'integrità del carico utile, mentre l'assenza di stato (*statelessness*) solleva il *server* dal mantenimento delle sessioni in memoria. Tale architettura favorisce una scalabilità orizzontale estrema.. i, 30

Large Language Model (LLM) Rete neurale profonda addestrata su immensi *corpus* testuali mediante l'architettura *Transformer*. Nell'ingegneria del *software*, gli LLM agiscono da assistenti cognitivi che deducono il contesto del progetto per generare frammenti di codice, automatizzare la documentazione e individuare vulnerabilità logiche.. i, 10

Machine Learning Sottocampo dell'intelligenza artificiale che sostituisce la programmazione esplicita con l'inferenza statistica. Gli algoritmi di apprendimento automatico assimilano schemi latenti a partire da vasti *dataset*, e ottimizzano progressivamente i propri parametri matematici per formulare previsioni accurate su dati inediti.. 4

Minimum Viable Product (MVP) Prototipo funzionale concepito per validare un'ipotesi di mercato con il minimo sforzo ingegneristico. A differenza di una dimostrazione puramente tecnica, l'MVP affronta un ciclo reale di utilizzo per raccogliere metriche empiriche e orientare gli sviluppi successivi verso i bisogni effettivi degli utenti.. ii, 18

Object-Relational Mapping (ORM) Livello di astrazione che risolve il disallineamento strutturale tra il paradigma orientato agli oggetti e le tabelle relazionali. Un ORM traduce fluidamente le istanze in *query* SQL ottimizzate, e offre strumenti nativi per la gestione delle transazioni, la concorrenza ottimistica e il caricamento differito (*lazy loading*) dei dati.. ii, 14

OLE Control Extension (OCX) Moduli *software* precompilati basati sulla tecnologia Component Object Model (COM) di Microsoft. Benché l'industria odierna li classifichi come *legacy*, numerosi ecosistemi gestionali storici vi fanno ancora affidamento per estendere le capacità delle interfacce grafiche in ambiente Windows.. ii, 2

One Time Password (OTP) Meccanismo crittografico per la generazione di credenziali monouso, spesso derivate tramite algoritmi basati sul tempo (TOTP) o su contatori sequenziali (HOTP). Rappresenta il cardine dell'autenticazione a più fattori, essenziale per neutralizzare le vulnerabilità delle *password* statiche e gli attacchi di intercettazione.. ii, 30

Project Management Disciplina organizzativa che orchestra risorse, tempi e costi per completare un progetto IT. I *framework* moderni abbandonano il rigore predittivo dei modelli a cascata in favore di approcci empirici e

adattivi, che massimizzano la consegna continua di valore e accorciano i cicli di *feedback* con gli *stakeholder*.. 22

Proof of Concept (PoC) Artefatto ingegneristico effimero che i *team* realizzano per validare la fattibilità tecnica di una specifica tecnologia. A differenza del Minimum Viable Product, il PoC non raggiunge l'utente finale: l'azienda scarta il suo codice sperimentale una volta estratte le metriche necessarie, per evitare l'accumulo di debito tecnico.. ii, 11

Rate Limiting Meccanismo di difesa infrastrutturale che disciplina il volume di traffico indirizzato a una risorsa di rete. L'implementazione di algoritmi quali il *Token Bucket* limita il tasso di richieste per singolo indirizzo IP o *token* utente, un approccio vitale per prevenire gli attacchi di negazione del servizio (DoS) e garantire la stabilità del sistema.. 34

Representational State Transfer (RESTful) Stile architetturale per i sistemi distribuiti che sfrutta pienamente i verbi e i codici di stato del protocollo HTTP. Impone vincoli strutturali netti, tra cui l'assenza di stato della sessione e l'interfaccia uniforme, per garantire il disaccoppiamento totale tra l'evoluzione del *client* e i meccanismi di persistenza del *server*.. ii, 14

Research and Development (R&D) Dipartimento nevralgico che le aziende tecnologiche impiegano per scongiurare l'obsolescenza dei propri prodotti. Le sue attività includono l'esplorazione dei *trend* architetturali emergenti e la prototipazione rapida di nuovi *framework*, per garantire la competitività dell'impresa in un mercato ad alta volatilità.. ii, 11

Scrum *Framework* metodologico per lo sviluppo *Agile* dei prodotti. Organizza il lavoro in iterazioni brevi (*Sprint*) e assegna responsabilità trasparenti a figure cardine: lo *Scrum Master* rimuove gli impedimenti procedurali, mentre il *Product Owner* massimizza il valore del prodotto e governa dinamicamente la lista delle priorità.. 5

SOLID Insieme di cinque postulati cardine (*Single Responsibility, Open-Closed, Liskov Substitution, Interface Segregation, Dependency Inversion*) ideati per la programmazione orientata agli oggetti. La loro rigorosa applicazione contrasta la rigidità del *software* e favorisce la creazione di architetture altamente modulari, estensibili e facili da sottoporre a *test*.. 7

System Design Disciplina ingegneristica che orchestra architetture su larga scala. I progettisti valutano continuamente i compromessi tra affidabilità, latenza e complessità, e prendono decisioni cruciali sul partizionamento dei *database*, sui protocolli di comunicazione e sulle strategie di replica, per soddisfare i vincoli del teorema CAP.. 19

Unified Modeling Language (UML) Linguaggio di modellazione visiva standardizzato per l'ingegneria del *software*. Le metodologie *Agile* contemporanee lo hanno spogliato della sua rigidità documentale per promuovere il paradigma dell'*UML as sketch*: i *team* lo impiegano oggi per tracciare diagrammi rapidi, utili a comunicare *pattern* strutturali senza l'onere di un mantenimento formale continuo.. ii, 23

User Experience (UX) Disciplina ingegneristica e psicologica che modella l'interazione tra l'utente e il prodotto digitale. Supera la pura progettazione visiva (*User Interface*) per abbracciare l'accessibilità, l'architettura dell'informazione e la fluidità dei percorsi cognitivi, con l'obiettivo di minimizzare la frustrazione operativa.. ii, 30

Version Control System (VCS) Infrastruttura crittografica e relazionale che traccia l'evoluzione di una *codebase*. I sistemi distribuiti contemporanei, come Git, memorizzano la cronologia in grafi aciclici diretti; tale struttura abilita la clonazione locale dei *repository*, la ramificazione indipendente del lavoro e la risoluzione sicura dei conflitti.. ii, 10

Web Content Accessibility Guidelines (WCAG) Linee guida internazionali pubblicate dal W3C per l'accessibilità dei contenuti *web*. La loro adozione impone l'impiego di HTML semantico, un contrasto cromatico

elevato e il supporto alla sintesi vocale. Il rispetto di questi standard estende la fruibilità degli applicativi alle persone con disabilità visive, uditive o motorie.. ii, 37