Università degli Studi di Salerno
Corso di Ingegneria del Software

SpeedScale
System Design Document
Versione 0.5

Data: 16/12/2024

Coordinatore del progetto:

Nome	Matricola
Sepe Gennaro	0512116971

Partecipanti:

Nome	Matricola
Sepe Gennaro	0512116971
La Marca Antonio	0512117826

Scritto da:	Sepe Gennaro

Revision History

Data	Versione	Descrizione	Autore
15/11/2024	0.1	Creazione del documento	La Marca Antonio
20/11/2024	0.2	Inizio della scrittura del documento	La Marca Antonio
24/11/2024	0.3	Completamento della scrittura	Sepe Gennaro
14/12/2024	0.4	Aggiustamenti post-consegna	Sepe Gennaro
16/12/2024	0.5	Ulteriori correzioni	Sepe Gennaro

Indice

1. Introduzione 4
1.1 Scopo del sistema 4
1.2 Obiettivi di progettazione 4
1.3 Definizioni, acronimi e abbreviazioni 6
1.4 Riferimenti 6
1.5 Panoramica 6
2. Architettura software attuale 6
3. Architettura software proposta 6
3.1 Panoramica 7
3.2 Scomposizione del sottosistema 7
3.3 Mappatura hardware/software 8
3.4 Gestione dei dati persistenti 9
3.5 Controllo degli accessi e sicurezza 9
3.6 Controllo globale del software 9
3.7 Condizioni di confine 10
4. Servizi del sottosistema 11
5. Glossario 11

1. Introduzione

1.1 Scopo del sistema

Il progetto SpeedScale nasce per rispondere alla crescente domanda, da parte del mercato italiano del modellismo automobilistico, di una piattaforma e-commerce altamente specializzata. Questo settore, ricco di appassionati e collezionisti, soffre attualmente la mancanza di un portale dedicato in grado di coniugare qualità, attenzione al dettaglio e facilità d'uso. Lo scopo del sistema è creare un'esperienza d'acquisto unica per gli amanti delle auto da corsa e delle riproduzioni storiche iconiche, offrendo un catalogo curato di modellini in scala. La piattaforma garantirà una presentazione accurata dei prodotti, con descrizioni dettagliate, immagini ad alta risoluzione e informazioni tecniche approfondite, per soddisfare una clientela esigente e orientata al dettaglio. Attraverso un'interfaccia intuitiva e piacevole, SpeedScale punta a semplificare la navigazione e il processo di acquisto, eliminando le complessità tipiche dei portali e-commerce non specializzati. L'obiettivo è posizionarsi come punto di riferimento per gli appassionati del settore, coniugando specializzazione e qualità in un contesto che valorizza sia i prodotti offerti sia l'esperienza utente complessiva.

1.2 Obiettivi di progettazione

1.2.1 Portabilità

Il sistema sarà distribuito tramite container Docker preconfigurati, assicurando portabilità e coerenza su diversi ambienti, semplificando il deployment e la manutenzione. Mentre per quanto riguarda il lato client, il sito sarà facilmente raggiungibile da tutti i browser maggiormente diffusi.

1.2.2 Robustezza

La piattaforma sarà progettata per gestire in modo efficace anomalie e guasti. Tutte le anomalie verranno registrate in log file dettagliati per agevolare la diagnosi, mentre gli utenti saranno informati in modo chiaro solo delle anomalie critiche, evitando frustrazioni. Tutti i dati saranno controllati sia lato client che lato server, così da garantire una prevenzione massima sia per possibili attacchi malevoli che per errori di sviluppo.

1.2.3 Facilità d'uso

L'interfaccia utente sarà ottimizzata per garantire un'esperienza semplice e piacevole. Suggerimenti, placeholder descrittivi e finestre in sovraimpressione accompagneranno gli utenti durante la compilazione dei moduli e l'interazione con il sistema, rendendo ogni azione chiara e priva di ambiguità.

1.2.4 Prestazioni elevate

Le prestazioni saranno garantire dalla gestione intelligente dei file multimediali più pesanti (come immagini e video) che include una compressione automatica prima del salvataggio e un reperimento degli stessi tramite richieste a un server separato e dedicato interamente alla loro persistenza.

1.2.5 Adattabilità

La piattaforma sarà progettata per supportare facilmente l'evoluzione delle esigenze di mercato. Ciò include la possibilità di aggiungere funzionalità multilingua e sistemi di ricerca avanzati con completamento automatico. In aggiunta, l'interfaccia del sito sarà completamente compatibile con tutte le risoluzioni più diffuse.

1.2.6 Affidabilità

La raggiungibilità del sistema sarà garantita al 99.5%, con finestre di manutenzione programmate

esclusivamente durante le ore notturne per minimizzare l'impatto sugli utenti. L'architettura scelta, basata su Java e MySQL, assicurerà stabilità e scalabilità nel tempo.

1.2.7 Tolleranza ai guasti
Il sistema sarà resiliente agli errori grazie a una progettazione che prevede un sistema automatico di backup con la possibilità di recuperare le funzionalità critiche in tempi ridotti, mitigando al minimo i disservizi.

1.2.8 Interfacce ben definite
L'interazione tra i moduli sarà garantita da interfacce chiaramente documentate, semplificando l'integrazione e il mantenimento del sistema. Saranno previste anche misure di sicurezza basate sui ruoli, così da impedire accessi non autorizzati a funzionalità riservate.

1.2.9 Semplicità d'uso
Ogni componente del sistema sarà sviluppato tenendo conto delle esigenze degli utenti finali, rispettando standard di accessibilità come le WCAG 2.1 AA, e ottimizzando la coerenza visiva tra le sezioni della piattaforma.

1.2.10 Efficienza
L'utilizzo di un database relazionale (MySQL) e di strumenti di version control (Git) garantirà processi rapidi e affidabili per la gestione dei dati e lo sviluppo collaborativo. Con queste tecnologie sarà possibile individuare esattamente dove un eventuale bug è stato introdotto in fase di sviluppo, così da poter procedere alla creazione di una patch risolutiva in tempi brevi. Inoltre, tutta la parte che si occuperà della connessione al database applicherà politiche di pooling e caching, così da minimizzare le istanziazioni delle risorse e gli accessi al servizio di storage.

1.2.11 Scalabilità
La piattaforma sarà progettata per gestire un numero crescente di utenti e prodotti. Grazie all'architettura modulare e a basso accoppiamento, sarà sempre possibile modificare o sostituire totalmente uno o più moduli senza alterarne i servizi offerti.

1.2.12 Sviluppo rapido
Grazie a strumenti moderni come il framework Bootstrap per il front-end e Git per il controllo delle versioni, sarà possibile mantenere un ritmo di sviluppo veloce senza compromettere la qualità del codice. Per quanto riguarda la prima versione del sistema, si preferirà implementare il salvataggio dei dati dell'utente nella sessione (da sincronizzare con il database).

1.2.13 Efficienza dei costi
La piattaforma sfrutterà risorse open-source e container Docker per minimizzare i costi di distribuzione e mantenimento, senza sacrificare qualità e prestazioni. Questa scelta si basa sul fatto che quasi tutti gli strumenti maggiormente diffusi nel settore e-commerce sono proprio open-source, ampiamente documentati e semplici da imparare in poco tempo grazie alle numerose risorse che possono essere trovate gratuitamente online.

1.2.14 Trade-off
- Affidabilità vs Costo-efficacia: La scelta di un'architettura robusta e di un uptime elevato richiederà un investimento iniziale maggiore, bilanciato però dalla riduzione dei costi operativi grazie all'uso di tecnologie efficienti. Si tenga pur sempre in considerazione che

l'affidabilità sarà sempre il primo requisito da soddisfare, anche se dovessero esserci aumenti di costo per l'acquisto di nuove infrastrutture.

- Scalabilità vs Sviluppo rapido: Sebbene lo sviluppo rapido sia prioritario, si è optato per un'architettura modulare che permetta di scalare facilmente nel tempo, accettando una possibile maggiore complessità iniziale nello sviluppo. Tale scelta si basa sulla volontà di consegnare in tempo il sistema, con la possibilità di integrare le componenti meno rilevanti in futuro tramite degli aggiornamenti.
- Usabilità vs Prestazioni: Se l'integrazione di elementi estetici (come effetti visivi ed animazioni) rallenta le prestazioni del sistema, allora si procederà con la loro eliminazione. Eventualmente potranno essere reintrodotte in futuro una volta ottimizzate.

1.3 Definizioni, acronimi e abbreviazioni

Termine	Definizione
Design goal	Obiettivi di design
Design trade-off	Compromessi tra obiettivi di design
COTS	Commercial Off-The-Shelf

1.4 Riferimenti

Per la progettazione di SpeedScale sono stati utilizzati come base teorica il manuale "Object Oriented Software Engineering Using UML, Patterns, and Java™" di Bernd Bruegge e Allen H. Dutoit, il quale ha offerto le linee guida per la scrittura dei documenti e l'uso di UML. Inoltre, è stato sviluppato precedentemente un "Problem Statement", che ha permesso di definire con precisione gli obiettivi e le esigenze del mercato. Successivamente è stato compilato un "Requirements Analysis Document", per poter iniziare ad avere una visione di insieme su una possibile strutturazione iniziale del sistema. Infine, il progetto è stato continuamente arricchito dagli insegnamenti del corso universitario "Ingegneria del Software" del professore De Lucia Andrea, che ha trattato le competenze per una gestione strutturata del ciclo di vita dello sviluppo software, applicate per garantire qualità e robustezza della piattaforma.

1.5 Panoramica

In questo capitolo sono state sviluppati gli argomenti riguardo la creazione del sistema, ponendo una maggiore enfasi sui design goals e i vari trade-off considerati. La parte seguente si concentrerà principalmente sull'individuazione dei sottosistemi, dell'architettura del sistema e delle procedure necessarie per gestirla.

2. Architettura software attuale

Il progetto parte da una concezione completamente nuova e non si basa su sistemi preesistenti. Al momento, non è disponibile alcuna architettura software operativa, né una piattaforma funzionante su cui basare l'evoluzione del sistema. Questa condizione consente al team di sviluppo di progettare una soluzione moderna e su misura, senza vincoli derivanti da strutture legacy o da tecnologie obsolete. Proprio grazie all'assenza di un sistema già esistente si presenta l'opportunità per integrare fin da subito le migliori pratiche di sviluppo, adottare tecnologie all'avanguardia e realizzare un'architettura che soddisfi pienamente gli obiettivi di progetto. Saranno prioritizzati design modulari e scalabili, che consentano di adattare il sistema alle future esigenze del mercato.

3. Architettura software proposta

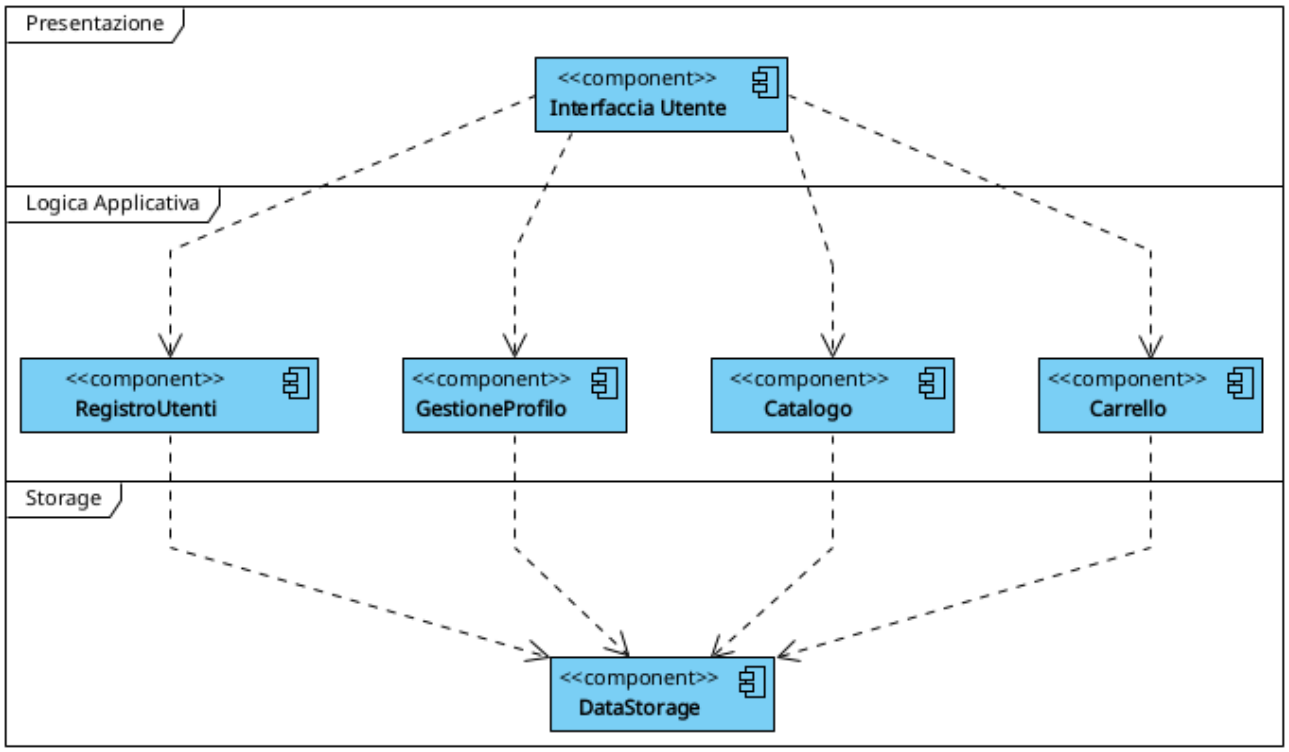
3.1 Panoramica

Il sistema sarà realizzato adottando un'architettura software three-tier, organizzata in tre livelli principali: interfaccia utente, logica applicativa e gestione dei dati. Questa scelta architetturale consente di ottenere una separazione chiara e modulare delle responsabilità, semplificando la manutenzione, favorendo la scalabilità e garantendo flessibilità nell'adattamento a future esigenze di mercato. Questa scelta è ritenuta ideale per rispondere alle caratteristiche di un sistema e-commerce moderno, dove l'interazione con l'utente, l'affidabilità delle operazioni e la sicurezza nella gestione dei dati sono fattori determinanti. Inoltre, grazie alla natura distribuita di questo modello, ogni layer può essere gestito e ottimizzato in modo indipendente, mantenendo alti livelli di performance e continuità del servizio.

Di seguito saranno descritti i dettagli dei tre strati dell'architettura:

1) **Strato di Interfaccia Utente (Presentation Layer):** Questo strato è dedicato alla gestione dell'interazione con l'utente finale, fornendo un'interfaccia web responsiva e intuitiva, progettata con Bootstrap per garantire compatibilità e una user experience di alto livello su dispositivi desktop e mobili. L'obiettivo è offrire una navigazione semplice, con funzionalità avanzate come suggerimenti di ricerca e un'interfaccia accessibile secondo gli standard WCAG 2.1 AA;
2) **Strato di Logica Applicativa (Application Layer):** La logica applicativa rappresenta il cuore della piattaforma, implementata in Java. Qui vengono gestite le principali funzionalità del sistema, come l'autenticazione degli utenti, il processo di acquisto, la gestione del carrello e delle preferenze personali. Questo strato è progettato per essere modulare e scalabile, garantendo un rapido adattamento a nuove funzionalità richieste dal mercato;
3) **Strato di Gestione dei Dati (Data Layer):** Il database relazionale MySQL costituisce il nucleo per l'archiviazione delle informazioni. Questo livello gestisce i dati relativi agli utenti, ai prodotti, agli ordini e alle attività di sistema, assicurando integrità, sicurezza e prestazioni elevate. Le tecniche di caching e pooling offerti dalla tecnica Object-Relational Mapping (ORM) riducono il carico sul database principale, migliorando i tempi di risposta e garantendo una maggiore efficienza.

3.2 Scomposizione del sottosistema

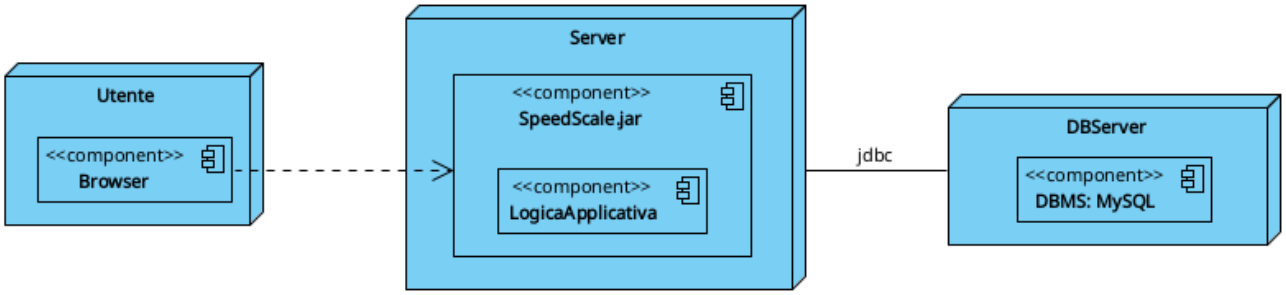


Di seguito le funzionalità di ogni sottosistema:

- **InterfacciaUtente**: Responsabile dell'interazione con l'utente e del coordinamento della logica applicativa tramite delegazione ai sottosistemi dello strato inferiore;
- **RegistroUtenti**: Responsabile dell'autenticazione e la registrazione dei clienti (inizializza la sessione una volta autenticato);
- **GestioneProfilo**: Responsabile della gestione dei dati personali, indirizzi di spedizione, metodi di pagamento e lo storico ordini;
- **Catalogo**: Responsabile della gestione dei prodotti;
- **Carrello**: Responsabile della gestione del carrello e della sua finalizzazione in ordine;
- **DataStorage**: Responsabile dell'immagazzinamento dei dati.

3.3 Mappatura hardware/software

Il sistema SpeedScale sarà sviluppato utilizzando Java per il codice back-end, garantendo robustezza, scalabilità e facilità di manutenzione. Per il front-end, verrà impiegato Bootstrap, un framework che permetterà di realizzare pagine web moderne, responsive e facilmente accessibili da dispositivi desktop e mobili. Il database relazionale MySQL sarà utilizzato per gestire i dati della piattaforma, inclusi utenti, prodotti e transazioni, assicurando integrità, velocità nelle operazioni di lettura e scrittura e una gestione efficiente delle informazioni. L'architettura three-tier del sistema, con il front-end separato dal back-end e dal database, permetterà una gestione modulare e facilmente scalabile, sia in ambienti on-premise che in futuro su soluzioni cloud, a seconda delle esigenze. Il sistema sarà inoltre progettato per supportare la programmazione distribuita, consentendo ai vari punti vendita di operare in modo indipendente ma interconnesso, ottimizzando la gestione delle risorse e garantendo una continuità operativa in tutta la rete. Proprio per questi motivi, è stato scelto di distribuire il sistema con un file JAR eseguibile, in quanto integra sia il codice applicativo che il web container. Questo approccio permette un deployment semplice, una gestione centralizzata e facilita la portabilità, garantendo anche scalabilità e facilità di manutenzione a lungo termine.



3.4 Gestione dei dati persistenti

Per la gestione dei dati persistenti, il sistema adotta un database relazionale come strategia principale di persistenza. Questa scelta è motivata dalla necessità di supportare query concorrenti e meccanismi di transazione. I database relazionali offrono anche il vantaggio di utilizzare il linguaggio SQL, che consente di definire e manipolare tabelle, supportando vincoli di integrità referenziale che assicurano la coerenza dei dati. Nel sistema, sarà necessario memorizzare anche immagini relative ai modelli e ai servizi. Tuttavia, poiché la memorizzazione di immagini in formato BLOB (Binary Large Object) all'interno del database potrebbe compromettere le prestazioni, queste immagini non verranno archiviate direttamente nel database. Invece, saranno salvate in un server separato, e il database memorizzerà la URL del file, piuttosto che il file stesso. Per la gestione della persistenza dei dati, sarà utilizzato MySQL come Database Management System (DBMS). Gli sviluppatori interagiranno con il database utilizzando JPA (Java Persistence API), una specifica Java che semplifica la gestione della persistenza dei dati e del mapping relazionale-oggetti. L'utilizzo di JPA consente di interagire con il database tramite oggetti Java, evitando la scrittura diretta di query SQL e migliorando la manutenibilità del codice.

3.5 Controllo degli accessi e sicurezza

I controlli della matrice degli accessi saranno gestiti tramite l'utilizzo delle annotazioni di sicurezza e dei ruoli definiti in Java. In particolare, saranno utilizzate le annotazioni come @PreAuthorize o @Secured per limitare l'accesso alle risorse in base ai ruoli dell'utente, garantendo che solo gli utenti autorizzati possano eseguire determinati operazioni. I ruoli, definiti all'interno del sistema, controlleranno le autorizzazioni su specifiche azioni, come la visualizzazione di determinati dati o l'esecuzione di operazioni sensibili.

Ogg/Att	**UteNonReg**	**UteReg**	**GesOrd**	**ResMag**	**GesPunVen**
RegistroUtenti	createUtente	authenticate	authenticate	authenticate	authenticate
Cliente		*all*	getOrdini		getOrdini
Carrello	*all*	*all*			
Prodotto				*all*	*all*
Catalogo				*all*	*all*
Ordine			*all*		*all*

N.B.
Con la dicitura "*all"* si intende che l'attore ha accesso a tutti le operazioni dell'oggetto.

3.6 Controllo globale del software

Considerato che i diagrammi di sequenza hanno dimostrato come il sistema abbia una struttura di controllo fortemente centralizzata, si decide di adottare un flusso di controllo basato su eventi, così da poter usufruire tutte le potenzialità del paradigma MVC (Model-View-Controller).
Entrando più nel dettaglio, il back-end riceve le richieste provenienti dal web browser del cliente, le elabora e le inoltra al controller. Questo processo crea un flusso di lavoro che risponde agli eventi

generati dall'utente. Ogni richiesta attiverà un nuovo thread, consentendo così una gestione concorrente delle richieste in parallelo, garantendo la fluidità e l'efficienza del sistema.
Gli oggetti di tipo boundary saranno progettati in modo da non mantenere stati persistenti, ma solo dati temporanei legati alla singola richiesta, evitando così conflitti. Poiché questi oggetti sono acceduti in maniera concorrente, l'assenza di campi persistenti elimina il rischio di problematiche legate alla sincronizzazione. Per quanto riguarda gli oggetti di controllo, essi non verranno condivisi tra i thread: ogni sessione avrà un unico oggetto di controllo, impedendo conflitti derivanti da richieste concorrenti sulla stessa risorsa. Gli oggetti entità saranno progettati per consentire l'accesso e la modifica dello stato solo attraverso metodi dedicati, garantendo un controllo preciso e sicuro sulle informazioni gestite. Nel contesto enterprise in cui il sistema opererà, la gestione delle transazioni e della concorrenza sarà delegata automaticamente al container, riducendo il carico di lavoro a livello di codice e aumentando l'affidabilità del sistema.

3.7 Condizioni di confine

Non saranno sviluppati componenti software aggiuntive per le fasi di inizializzazione, terminazione e fallimento del sistema. Quest'ultime verranno gestite dalla figura del "system administrator" (che potrebbe coincidere con il gestore del punto vendita). Questa scelta si basa sulle considerazioni sviluppate precedentemente, in cui si afferma che il sistema può essere facilmente messo in produzione facendo partire un container docker senza configurazioni aggiuntive necessarie per il suo funzionamento. Eventualmente sarà possibile accedere alle sezioni di amministrazione dei componenti del web container tramite le loro interfacce web raggiungibili con un qualsiasi browser.

4. Servizi del sottosistema

RegistroUtenti	
Autenticazione	authenticate(String email, String pwd)
	inizializzaSessione(Utente utente)
Registrazione	createUtente(String email, String pwd, String nome, String cognome, Date dataNascita)

GestioneProfilo	
Gestione Dati Personali	modificaDatiPersonali(Utente utente, String newNome, String newCognome, String newPassword, Date newDataNascita)
Gestione Indirizzi Spedizione	addIndirizzoSpedizione(Utente utente, Indirizzo newIndirizzo)
	modificaIndirizzoSpedizione(Utente utente, Indirizzo indirizzo, Indirizzo newIndirizzo)
	removeIndirizzoSpedizione(Utente utente, Indirizzo indirizzo)
Gestione Metodi Pagamento	aggiungiMetodoPagamento(Utente utente, MetodoPagamento newMetodoPagamento)
	modificaMetodoPagamento(Utente utente, MetodoPagameno metodoPagamento, MetodoPagamento newMetodoPagamento)
	removeMetodoPagamento(Utente utente, MetodoPagamento metodoPagamento)
Gestione Ordini	visualizzaStoricoOrdini(Utente utente)

Catalogo	
Gestione Prodotto	addNewProdotto(Prodotto prodotto)
	retrieveProdotto(Long prodottoId)
	modificaProdotto(Prodotto prodotto, Prodotto nuovoProdotto)
	rimuoviProdotto(Prodotto prodotto)
	getProdotti()

Carrello	
Gestione Carrello	addProdottoCarrello(Utente utente, Prodotto prodotto, int quantita)
	removeProdottoCarrello(Utente utente, Prodotto prodotto)
	UpdateQuantitaProdotto(Utente utente, Prodotto prodotto, int newQuantita)
	getCarrello(Utente utente)
Check Out	creaOrdine(Utente utente, Indirizzo indirizzo, MetodoPagamento pagamento, Carrello carrello)

5. Glossario

Non sono stati introdotte nuove terminologie particolari.