Università degli Studi di Salerno
Corso di Ingegneria del Software

GINus Juice
Object Design Document

Versione 1.0

Data: 10/11/2024

Coordinatore del progetto:

Nome	Matricola
ANTONIO VITULANO	0512116776
SALVATORE D'AVINO	0512118435

Partecipanti:

Nome	Matricola
ANTONIO VITULANO	0512116776
SALVATORE D'AVINO	0512118435

Scritto da:	ANTONIO VITULANO, SALVATORE D'AVINO

Revision History

Data	Versione	Descrizione	Autore
16/12/2024	**1.0**	**Definizioni principali del documento**	**AV,SD**
29/12/2024	1.0	Aggiunta dettagli	AV,SD
05/12/24	1.1	Affinamento delle specifiche delle classi	AV,SD

Sommario

1. Introduction

Dopo la realizzazione dei documenti RAD e SDD abbiamo descritto in linea massima quello che sarà il nostro sistema e quindi i nostri obiettivi, tralasciando gli aspetti d'implementazione.

Il seguente documento ha lo scopo di produrre un modello capace di integrare in modo coerente e preciso tutte le diverse funzionalità individuate nelle fasi precedenti.

In particolare, si vanno a descrivere i trade-offs che dovranno essere rispettati dagli sviluppatori, i design pattern stabiliti, le linee guida sulla documentazione delle interfacce, la divisione in pacchetti e le interfacce delle classi da sviluppare:

1.1 Object design trade-offs

Trade-Off	Motivazione

Comprensibilità vs Tempo	Il codice del sistema deve essere comprensibile, in modo da facilitare la fase di testing ed eventuali future modifiche da apportare. Al fine di rispettare queste linee guida il codice sarà integrato da commenti volti a migliorarne la leggibilità; tuttavia questo richiederà una maggiore quantità di tempo necessario per lo sviluppo del nostro progetto.
Manutenibilità vs. Performance	Il sistema sarà progettato in modo tale da permettere, in futuro e quando se ne necessiterà, una semplice manutenzione. Per ottenere ciò si dovrà prediligere una scrittura del codice volta alla modularità delle sue parti, a discapito della pratica dell'hard coding.
Sicurezza vs Costi	La sicurezza rappresenta uno degli aspetti principali del sistema. Tuttavia, a causa di tempi di sviluppo molto limitati, ci limiteremo ad implementare sistemi di sicurezza basati su email e password.
Funzionalità vs Usabilità	Verrà realizzata un'interfaccia grafica chiara e concisa, usando form e pulsanti predefiniti che hanno lo scopo di rendere semplice l'utilizzo del sistema da parte dell'utente finale, delegando a pagine specifiche le funzionalità più avanzate che minano l'uso intuitivo dell'applicazione.
Sviluppo rapido vs Features	Le funzionalità specifiche dell'applicazione verranno realizzate seguendo un sistema basato su delle priorità. Privilegiando uno sviluppo rapido, verrà data la precedenza agli elementi che dispongono di una priorità alta per poi integrare le restanti funzionalità in un secondo momento.

Di seguito è riportata una tabella che mostra i design goal preferiti nei Trade Off. Il grassetto indica la preferenza.

Trade-Off	
Performance	**Memoria**
Affidabilità	**Tempo di risposta**
Disponibilità	**Tolleranza ai guasti**
Manutenibilità	**Performance**

1.1.2 DESIGN PATTERN

Singleton: Il Singleton è un design pattern che garantisce l'esistenza di una sola istanza di una classe in tutta l'applicazione, fornendo un accesso globale a tale istanza. È utile per gestire risorse condivise, come configurazioni o connessioni. Nel nostro caso la abbiamo utilizzato per gestire i prodotti

1.1.2.2 **Design Pattern DAO**

Il DAO (Data Access Object) è un pattern architetturale per la gestione della persistenza: si tratta fondamentalmente di una classe con relativi metodi che rappresenta un'entità tabellare di un RDBMS, usata principalmente in applicazioni web per stratificare e isolare l'accesso ad una tabella tramite Query

I metodi DAO contenenti le varie query verranno così richiamati dalle classi della bussiness logic

1.2 Interface documentation guidelines

E richiesto agli sviluppatori di seguire le seguenti linee guida al fine di essereconsistenti nell'intero progetto e facilitare la comprensione delle funzionalità di ogni componente.

1.2.1 Classi e Interfacce Java

Lo standard nella definizione delle classi e delle interfacce Java è quello definito da Google. Ciascuna classe e ciascun metodo deve essere documentato seguendo lo stile di documentazione JavaDoc.

1.2.2 Java Servlet Pages (JSP)

Le JSP costruiscono pagine HTML in maniera dinamica che devono rispettare il formato definito nel sotto paragrafo sottostante. Devono anche attenersi alle linee guida per le classi Java definito nel sotto paragrafo soprastante. Inoltre, le JSP devono attenersi alle seguenti puntualizzazioni:

1. Il tag di apertura (<%) è seguito immediatamente dalla fine della riga;

2. Il tag di chiusura (%>) si trova all'inizio di una riga che non contiene nient'altro;

3. Istruzioni Java che consistono in una sola riga possono contenere il tag di apertura e chiusura sulla stessa riga;

1.2.3 Pagine HTML

Le pagine HTML devono essere conformi allo standard HTML5. Inoltre, il codice HTML deve utilizzare l'indentazione per facilitare la lettura e di conseguenza la manutenzione. Le regole di indentazione sono le seguenti:

1. Ogni tag deve avere un'indentazione maggiore del tag che lo contiene;

2. Ogni tag di chiusura deve avere lo stesso livello di indentazione di quello di apertura;

1.2.4 Fogli di stile CSS

Ogni foglio di stile deve essere inizializzato da un commento analogo a quello presente nei file Java.

Ogni regola CSS deve essere formattata nel seguente modo:

1. I selettori della regola si trovano a livello 0 di indentazione, uno per riga;

2. L'ultimo selettore della regola è seguito da parentesi graffa aperta ({);

3. Le proprietà che costituiscono la regola sono listate una per riga e sono indentate rispetto ai

selettori;

4. La regola è terminata da una parentesi graffa chiusa (}), collocata da sola su una riga;

1.2.5 Script SQL

Le istruzioni e le clausole SQL devono essere costituite da sole lettere maiuscole. I nomi delle tabelle hanno la prima lettera maiuscola e le successive minuscole e il loro nome deve essere un sostantivo singolare. I nomi degli attributi devono essere costituiti da sole lettere minuscole e possono contenere più parole separate da un underscore (_).

1.3 Definitions, acronyms, and abbreviations

Definizioni

DOM = Modella la struttura di pagine web.

MVC: Model-view-controller è un pattern architetturale molto diffuso nello sviluppo di sistemi software.

DAO (Data Access Object): è un pattern architetturale per la gestione della persistenza.

Acronimi

ODD = Object Design Document

GUI = Graphical User Interface (Interfaccia grafica utente)

HTML = HyperText Markup Language

CSS = Cascading Style Sheet

DOM = Document Object Model

SQL = Structured Query Language

JSP = Java Servlet Pages

1.4 References

Problem statement Rad,Sdd

2. Packages

INTRODUZIONE

il sistema sia (web app che applicazione desktop) rispecchia l'architettura definita nel documento SDD, quindi avremo un sistema suddiviso in tre livelli (three-tier):

2.1 Divisione in pacchetti

In questa sezione presentiamo in modo più approfondito quella che è la divisione in sottosistemi e l'organizzazione del codice in file. Il sistema sia rispecchia l'architettura definita nel documento SDD, quindi avremo un sistema suddiviso in tre livelli (three-tier):

Presentazione Layer	Rappresenta l'interfaccia del sistema, ed offre la possibilità all'utente di interagire con quest'ultimo, offrendo sia la possibilità di inviare, in input, che di visualizzare in output dati.

Business Layer	Ha il compito di elaborare i dati da inviare al client, e spesso grazie a delle richieste fatte al database, tramite il Persistence Layer, accede ai dati persistenti. Si occupa di varie gestioni quali: • Accesso • Prenotazione • Visualizzazione Coda • Gestione
Persistence Layer	Ha il compito di memorizzare i dati sensibili del sistema utilizzando un DBMS, inoltre riceve le varie richieste dal Business Layer inoltrandole al DBMS e restituendo i dati richiesti.

3. Class interfaces

Class Interface Tables

Tabelle delle Specifiche delle Interfacce - Model

Acquisto

Nome	**Acquisto**
Descrizione	Rappresenta un acquisto effettuato da un utente.
Attributi	- idAcquisto: int - dataAcquisto: Date - prezzoTotale: double- emailUtente: String
Signature dei metodi	+ getIdAcquisto() : int+

	setIdAcquisto(int idAcquisto) : void + getDataAcquisto() : Date + setDataAcquisto(Date dataAcquisto) : void + getPrezzoTotale() : double + setPrezzoTotale(double prezzoTotale) : void+ getEmailUtente() : String+ setEmailUtente(String emailUtente) : void
Pre-condizioni	L'acquisto deve essere associato a un utente valido.
Post-condizioni	I dettagli dell'acquisto vengono registrati correttamente.
Invarianti	Nessuna

AcquistoDAO

Nome	AcquistoDAO
Descrizione	Gestisce l'accesso ai dati relativi agli acquisti effettuati dagli utenti.
Attributi	Nessuno
Signature dei metodi	+ doSave(Acquisto acquisto) : void + doRetrieveById(int idAcquisto) : Acquisto + doRetrieveByUser(String emailUtente) : List + doDelete(int idAcquisto) : void
Pre-condizioni	Il database deve essere accessibile.
Post-condizioni	Le operazioni sugli acquisti vengono eseguite correttamente.
Invarianti	Nessuna

Carrello

Nome	Carrello

Descrizione	Gestisce il carrello di un utente, permettendo l'aggiunta e rimozione di prodotti.
Attributi	- prodotti: Map<Prodotto, Integer>
Signature dei metodi	+ aggiungiProdotto(Prodotto prodotto, int quantita) : void + rimuoviProdotto(Prodotto prodotto) : void + getTotale() : double
Pre-condizioni	I prodotti aggiunti al carrello devono essere disponibili in magazzino.
Post-condizioni	Il carrello viene aggiornato correttamente con i prodotti e le quantità richieste.
Invarianti	Nessuna

ConnectionPool

Nome	ConnectionPool
Descrizione	Gestisce il pool di connessioni al database.
Attributi	- instance: ConnectionPoo l- connections: Queue
Signature dei metodi	+ getInstance() : ConnectionPool + getConnection() : Connection + releaseConnection(Connection conn) : void
Pre-condizioni	Il database deve essere configurato correttamente.
Post-condizioni	Le connessioni vengono allocate e rilasciate correttamente.
Invarianti	La dimensione del pool deve rimanere entro i limiti definiti.

Domanda

Nome	Domanda
Descrizione	Rappresenta una domanda effettuata da un utente.
Attributi	- idDomanda: int- corpoDomanda: String - dataDomanda: Date - emailUtente: String
Signature dei metodi	+ getIdDomanda() : int + setIdDomanda(int idDomanda) : void+ getCorpoDomanda() : String + setCorpoDomanda(String corpoDomanda) : void + getDataDomanda() : Date + setDataDomanda(Date dataDomanda) : void + getEmailUtente() : String + setEmailUtente(String emailUtente) : void
Pre-condizioni	I campi della domanda non devono essere vuoti.
Post-condizioni	La domanda viene registrata con successo.
Invarianti	Nessuna

DomandaDAO

Nome	DomandaDAO
Descrizione	Gestisce l'accesso ai dati relativi alle domande.
Attributi	Nessuno
Signature dei metodi	+ doSave(Domanda domanda) : void + doRetrieveById(int idDomanda) : Domanda + doDelete(int idDomanda) : void

	+ doRetrieveAll() : List
Pre-condizioni	Il database deve essere accessibile.
Post-condizioni	Le operazioni CRUD sulle domande vengono eseguite correttamente.
Invarianti	Nessuna

Prodotto

Nome	Prodotto
Descrizione	Rappresenta un prodotto disponibile nel sistema.
Attributi	- idProdotto: int - nome: String - descrizione: String - quantita: int - prezzo: double
Signature dei metodi	+ getIdProdotto() : int + setIdProdotto(int idProdotto) : void + getNome() : String + setNome(String nome) : void + getDescrizione() : String + setDescrizione(String descrizione) : void + getQuantita() : int + setQuantita(int quantita) : void + getPrezzo() : double + setPrezzo(double prezzo) : void
Pre-condizioni	I dati del prodotto devono rispettare i vincoli definiti (ad esempio, prezzo maggiore di zero).
Post-condizioni	Gli attributi del prodotto vengono impostati correttamente.

Invarianti	Nessuna

ProdottoDAO

Nome	ProdottoDAO
Descrizione	Gestisce l'accesso ai dati relativi ai prodotti.
Attributi	Nessuno
Signature dei metodi	+ doSave(Prodotto prodotto) : void + doRetrieveById(int idProdotto) : Prodotto + doDelete(int idProdotto) : void + doRetrieveAll() : List
Pre-condizioni	Il database deve essere accessibile.
Post-condizioni	Le operazioni CRUD sui prodotti vengono eseguite correttamente.
Invarianti	Nessuna

Recensione

Nome	Recensione
Descrizione	Modella una recensione effettuata da un utente su un prodotto.
Attributi	- idRecensione: int - corpoRecensione: String - dataRecensione: Date - idProdotto: int - emailUtente: String
Signature dei metodi	+ getIdRecensione() : int + setIdRecensione(int idRecensione) : void

	+ getCorpoRecensione() : String + setCorpoRecensione(String corpoRecensione) : void + getDataRecensione() : Date + setDataRecensione(Date dataRecensione) : void + getIdProdotto() : int + setIdProdotto(int idProdotto) : void + getEmailUtente() : String + setEmailUtente(String emailUtente) : void
Pre-condizioni	I dati devono rispettare i vincoli di validità definiti per ogni attributo (es. formato email corretto).
Post-condizioni	Gli attributi della recensione vengono impostati correttamente.
Invarianti	Nessuna

RecensioneDAO

Nome	RecensioneDAO
Descrizione	Gestisce l'accesso ai dati relativi alle recensioni.
Attributi	Nessuno
Signature dei metodi	+ doSave(Recensione recensione) : void + doRetrieveById(int idRecensione) : Recensione + doDelete(int idRecensione) : void + doRetrieveAll() : List
Pre-condizioni	Il database deve essere accessibile.
Post-condizioni	Le operazioni CRUD sulle recensioni vengono eseguite correttamente.
Invarianti	Nessuna

Risposta

Nome	Risposta
Descrizione	Rappresenta una risposta fornita a una domanda.
Attributi	- idRisposta: int - corpoRisposta: String - dataRisposta: Date - idDomanda: int
Signature dei metodi	+ getIdRisposta() : int + setIdRisposta(int idRisposta) : void + getCorpoRisposta() : String + setCorpoRisposta(String corpoRisposta) : void + getDataRisposta() : Date + setDataRisposta(Date dataRisposta) : void + getIdDomanda() : int + setIdDomanda(int idDomanda) : void
Pre-condizioni	La domanda associata deve esistere.
Post-condizioni	La risposta viene registrata con successo.
Invarianti	Nessuna

RispostaDAO

Nome	RispostaDAO
Descrizione	Gestisce l'accesso ai dati relativi alle risposte.
Attributi	Nessuno
Signature dei metodi	+ doSave(Risposta risposta) : void + doRetrieveById(int idRisposta) : Risposta + doDelete(int idRisposta) : void

	+ doRetrieveAll() : List
Pre-condizioni	Il database deve essere accessibile.
Post-condizioni	Le operazioni CRUD sulle risposte vengono eseguite correttamente.
Invarianti	Nessuna

Utente

Nome	Utente
Descrizione	Rappresenta un utente registrato nel sistema.
Attributi	- email: String - nome: String - cognome: String - password: String - ruolo: String - stato: boolean
Signature dei metodi	+ getEmail() : String + setEmail(String email) : void + getNome() : String + setNome(String nome) : void + getCognome() : String + setCognome(String cognome) : void + getPassword() : String + setPassword(String password) : void + getRuolo() : String + setRuolo(String ruolo) : void + isStato() : boolean + setStato(boolean stato) : void

Pre-condizioni	I campi non possono essere vuoti e devono rispettare i vincoli di validità.
Post-condizioni	L'utente viene configurato correttamente.
Invarianti	Nessuna

UtenteDAO

Nome	UtenteDAO
Descrizione	Gestisce l'accesso ai dati relativi agli utenti.
Attributi	Nessuno
Signature dei metodi	+ doSave(Utente utente) : void + doRetrieveByEmail(String email) : Utente + doDelete(String email) : void + doRetrieveAll() : List
Pre-condizioni	Il database deve essere accessibile e la connessione valida.
Post-condizioni	Le operazioni CRUD sugli utenti vengono eseguite correttamente.
Invarianti	Nessuna

Tabelle delle Specifiche delle Interfacce - Acquisto

AcquistoServlet

Nome	AcquistoServlet
Descrizione	Gestisce le richieste relative agli acquisti effettuati dagli utenti.
Attributi	Nessuno
Signature dei metodi	+ doGet(HttpServletRequest request, HttpServletResponse response) : void

	+ doPost(HttpServletRequest request, HttpServletResponse response) : void
Pre-condizioni	L'utente deve essere autenticato per poter effettuare o visualizzare acquisti.
Post-condizioni	I dettagli dell'acquisto vengono elaborati o visualizzati correttamente in base alla richiesta effettuata.
Invarianti	Nessuna

Tabelle delle Specifiche delle Interfacce - Autenticazione

RegistrazioneControl

Nome	**RegistrazioneControl**
Descrizione	Gestisce il processo di registrazione di un nuovo utente nel sistema.
Attributi	Nessuno
Signature dei metodi	+ doPost(HttpServletRequest request, HttpServletResponse response) : void
Pre-condizioni	Tutti i campi richiesti per la registrazione (ad esempio email, password, ecc.) devono essere forniti correttamente e rispettare i vincoli di validità.
Post-condizioni	Un nuovo utente viene creato nel sistema e i suoi dettagli vengono salvati nel database.
Invarianti	Nessuna

Logout

Nome	Logout
Descrizione	Gestisce il processo di logout per un utente autenticato.
Attributi	Nessuno
Signature dei metodi	+ doGet(HttpServletRequest request, HttpServletResponse response) : void
Pre-condizioni	L'utente deve essere autenticato e avere una sessione attiva.
Post-condizioni	La sessione dell'utente viene terminata e l'utente viene reindirizzato alla pagina di login o alla home page pubblica.
Invarianti	Nessuna

CheckLogin

Nome	CheckLogin
Descrizione	Gestisce la verifica delle credenziali fornite da un utente durante il processo di login.
Attributi	Nessuno
Signature dei metodi	+ doPost(HttpServletRequest request, HttpServletResponse response) : void
Pre-condizioni	L'utente deve fornire credenziali valide (email e password) e rispettare i vincoli di validità.
Post-condizioni	L'utente viene autenticato correttamente e reindirizzato alla pagina appropriata (dashboard o home page), oppure viene notificato un errore in caso di credenziali non valide.
Invarianti	Nessuna

Tabelle delle Specifiche delle Interfacce - Admin

AggiornaRuoloServlet

Nome	AggiornaRuoloServlet
Descrizione	Gestisce la modifica del ruolo di un utente da parte di un amministratore.
Attributi	Nessuno
Signature dei metodi	+ doPost(HttpServletRequest request, HttpServletResponse response) : void
Pre-condizioni	L'utente deve essere autenticato come amministratore e deve fornire i dati corretti per l'utente e il ruolo da aggiornare.
Post-condizioni	Il ruolo dell'utente specificato viene aggiornato correttamente nel sistema.
Invarianti	Nessuna

RimuoviUtenteServlet

Nome	RimuoviUtenteServlet
Descrizione	Gestisce la rimozione di un utente dal sistema da parte di un amministratore.

Attributi	Nessuno
Signature dei metodi	+ doPost(HttpServletRequest request, HttpServletResponse response) : void
Pre-condizioni	L'utente deve essere autenticato come amministratore e deve specificare correttamente l'utente da rimuovere.
Post-condizioni	L'utente specificato viene rimosso dal sistema, inclusi i dati associati.
Invarianti	Nessuna

StoricoAcquistiServlet

Nome	StoricoAcquistiServlet
Descrizione	Permette a un amministratore di visualizzare lo storico degli acquisti di un utente.
Attributi	Nessuno
Signature dei metodi	+ doGet(HttpServletRequest request, HttpServletResponse response) : void
Pre-condizioni	L'utente deve essere autenticato come amministratore e deve fornire i dettagli necessari per identificare l'utente di cui si vuole visualizzare lo storico acquisti.
Post-condizioni	Lo storico degli acquisti viene recuperato e presentato correttamente.
Invarianti	Nessuna

Tabelle delle Specifiche delle Interfacce - Carrello

CarrelloServlet

Nome	**CarrelloServlet**
Descrizione	Gestisce le operazioni principali relative al carrello di un utente, come aggiunta e visualizzazione dei prodotti.
Attributi	Nessuno
Signature dei metodi	+ doGet(HttpServletRequest request, HttpServletResponse response) : void + doPost(HttpServletRequest request, HttpServletResponse response) : void
Pre-condizioni	L'utente deve essere autenticato e deve avere un carrello associato.

Post-condizioni	Gli elementi del carrello vengono gestiti correttamente (aggiunti o visualizzati) in base alla richiesta effettuata.
Invarianti	Nessuna

SvuotaCarrelloServlet

Nome	SvuotaCarrelloServlet
Descrizione	Permette di svuotare completamente il carrello di un utente.
Attributi	Nessuno
Signature dei metodi	+ doPost(HttpServletRequest request, HttpServletResponse response) : void
Pre-condizioni	L'utente deve essere autenticato e deve avere un carrello con almeno un prodotto.
Post-condizioni	Il carrello dell'utente viene svuotato correttamente, rimuovendo tutti gli articoli presenti.
Invarianti	Nessuna

Tabelle delle Specifiche delle Interfacce – Chiedi al barista
VisualizzaRisposteServlet

Nome	**VisualizzaRisposteServlet**
Descrizione	Permette agli utenti di visualizzare le risposte fornite dal barista alle domande precedentemente inviate.
Attributi	Nessuno
Signature dei metodi	+ doGet(HttpServletRequest request, HttpServletResponse response) : void
Pre-condizioni	L'utente deve essere autenticato per accedere a questa funzionalità.
Post-condizioni	Le risposte associate all'utente vengono recuperate e visualizzate correttamente.
Invarianti	Nessuna

VisualizzaDomandeServlet

Nome	VisualizzaDomandeServlet
Descrizione	Permette agli utenti o al barista di visualizzare tutte le domande inviate dagli utenti.
Attributi	Nessuno

Signature dei metodi	+ doGet(HttpServletRequest request, HttpServletResponse response) : void
Pre-condizioni	L'utente deve essere autenticato per accedere a questa funzionalità.
Post-condizioni	Le domande inviate dagli utenti vengono recuperate e visualizzate correttamente.
Invarianti	Nessuna

RispondiDomandaServlet

Nome	RispondiDomandaServlet
Descrizione	Permette al barista di fornire una risposta a una domanda inviata da un utente.
Attributi	Nessuno
Signature dei metodi	+ doPost(HttpServletRequest request, HttpServletResponse response) : void
Pre-condizioni	L'utente deve essere autenticato come barista per accedere a questa funzionalità.
Post-condizioni	La risposta viene registrata correttamente nel sistema ed è associata alla domanda specificata.
Invarianti	Nessuna

ScriviDomandaServlet

Nome	ScriviDomandaServlet
Descrizione	Permette agli utenti di inviare una domanda al barista tramite il sistema.
Attributi	Nessuno
Signature dei metodi	+ doPost(HttpServletRequest request, HttpServletResponse response) : void
Pre-condizioni	L'utente deve essere autenticato e deve fornire un corpo domanda valido.
Post-condizioni	La domanda viene registrata correttamente nel sistema ed è associata all'utente che l'ha inviata.
Invarianti	Nessuna

Tabelle delle Specifiche delle Interfacce – Shop

ProdottoSingleton

Nome	ProdottoSingleton
Descrizione	Implementa il pattern Singleton per la gestione di un'istanza unica di Prodotto.
Attributi	- instance: ProdottoSingleton - prodotto: Prodotto
Signature dei metodi	+ getInstance() : ProdottoSingleton+ getProdotto() : Prodotto + setProdotto(Prodotto prodotto) : void
Pre-condizioni	L'istanza deve essere unica per tutto il sistema.
Post-condizioni	Garantisce un'unica istanza condivisa e permette di accedere o aggiornare il prodotto gestito.
Invarianti	Nessuna

AppInitializer

Nome	AppInitializer
Descrizione	Inizializza i parametri e le configurazioni necessarie durante l'avvio dell'applicazione.
Attributi	Nessuno
Signature dei metodi	+ contextInitialized(ServletContextEvent sce) : void + contextDestroyed(ServletContextEvent sce) : void
Pre-condizioni	Il contesto Servlet deve essere disponibile durante l'inizializzazione.
Post-condizioni	I parametri di configurazione vengono impostati correttamente e le risorse vengono liberate allo spegnimento.
Invarianti	Nessuna

VisualizzaDettagliServlet

Nome	VisualizzaDettagliServlet
Descrizione	Gestisce la visualizzazione dei dettagli di un prodotto specifico.
Attributi	Nessuno
Signature dei metodi	+ doGet(HttpServletRequest request, HttpServletResponse response) : void
Pre-condizioni	Deve essere specificato un identificativo valido per il prodotto.
Post-condizioni	I dettagli del prodotto vengono recuperati e presentati correttamente.
Invarianti	Nessuna

AggiungiRecensioneServlet

Nome	AggiungiRecensioneServlet
Descrizione	Permette agli utenti di aggiungere una recensione per un prodotto specifico.
Attributi	Nessuno
Signature dei metodi	+ doPost(HttpServletRequest request, HttpServletResponse response) : void
Pre-condizioni	L'utente deve essere autenticato e deve fornire una recensione valida (testo, valutazione, ecc.).
Post-condizioni	La recensione viene salvata nel sistema ed è associata al prodotto specificato.
Invarianti	Nessuna

Tabelle delle Specifiche delle Interfacce – Magazzino

EliminaProdottoServlet

Nome	**EliminaProdottoServlet**
Descrizione	Gestisce l'eliminazione di un prodotto dal magazzino.
Attributi	Nessuno
Signature dei metodi	+ doPost(HttpServletRequest request, HttpServletResponse response) : void
Pre-condizioni	Deve essere fornito un identificativo valido per il prodotto da eliminare.
Post-condizioni	Il prodotto specificato viene eliminato dal database del magazzino.
Invarianti	Nessuna

ModificaProdottoServlet

Nome	ModificaProdottoServlet
Descrizione	Permette di modificare i dettagli di un prodotto esistente nel magazzino.
Attributi	Nessuno

Signature dei metodi	+ doPost(HttpServletRequest request, HttpServletResponse response) : void
Pre-condizioni	Devono essere forniti dettagli validi e completi per il prodotto da modificare.
Post-condizioni	Le modifiche ai dettagli del prodotto vengono salvate correttamente nel database.
Invarianti	Nessuna

VisualizzaModificaProdottoServlet

Nome	VisualizzaModificaProdottoServlet
Descrizione	Gestisce la visualizzazione della pagina per modificare un prodotto specifico.
Attributi	Nessuno
Signature dei metodi	+ doGet(HttpServletRequest request, HttpServletResponse response) : void
Pre-condizioni	Deve essere specificato un identificativo valido per il prodotto da modificare.
Post-condizioni	La pagina di modifica del prodotto viene visualizzata con i dettagli attuali del prodotto specificato.
Invarianti	Nessuna

AggiungiProdottoServlet

Nome	AggiungiProdottoServlet
Descrizione	Permette di aggiungere un nuovo prodotto al magazzino.
Attributi	Nessuno
Signature dei metodi	+ doPost(HttpServletRequest request, HttpServletResponse response) : void
Pre-condizioni	Devono essere forniti dettagli validi e completi per il nuovo prodotto.
Post-condizioni	Il nuovo prodotto viene aggiunto correttamente al database del magazzino.
Invarianti	Nessuna