ODD: OBJECT DESIGN DOCUMENT

C05

DressMeNow

Riferimento	
Versione	1.0
Data	28/11/2023
Destinatario	Prof.ssa F. Ferrucci
Presentato da	Luigi Allocca, Luigi Emanuele Sica
Approvato da	

Revision History

Data	Versione	Descrizione	Autori
28/11/2023	0.1	Creazione	Tutti i membri del team
30/11/2023	0.2	Aggiornamento	Tutti i membri del team
13/12/2023	0.3	Revisione	Tutti i membri del team
13/12/2023	1.0	Prima stesura completa	Tutti i membri del team
14/12/2023	1.1	Revisione	Tutti i membri del team
18/12/2023	1.2	Aggiornamento	Tutti i membri del team
22/12/2023	1.3	Aggiornamento	Tutti i membri del team
22/12/2023	1.4	Revisione	Carlo di Vico
26/12/2023	1.5	Aggiornamento	Tutti i membri del team

Sommario

Sommario (Tutti i membri del team)

L'Object Design Document permette di dettagliare con precisione le scelte effettuate durante l'analisi e la progettazione. Verranno delineati i compromessi principali, descritte le componenti pronte all'uso integrate nel sistema, fornite linee guida per la documentazione delle interfacce e individuati i modelli architetturali. Infine, saranno definiti i pacchetti, le interfacce delle classi e i diagrammi che rappresentano le importanti decisioni di implementazione.

Object design trade-offs (Tutti i membri del team)

Utilizzo di memoria Vs Velocità di risposta

Dato che gli utenti devono usufruire di un'esperienza rapida e reattiva, si è deciso di attribuire maggiore importanza alla velocità di risposta rispetto all'utilizzazione della memoria.

Facilità d'uso Vs Costi, Prestazioni

Il sito sarà utilizzato da utenti aventi diversi livelli di esperienza di navigazione e di comprensione delle nuove tecnologie, pertanto, la chiarezza e la facilità d'uso sono fondamentali. Si darà priorità alla comprensibilità dell'interfaccia dell'avatar e dell'applicazione, anche se ciò richiede un aumento dei costi e potrebbe incidere leggermente sulle prestazioni.

Sicurezza Vs Personalizzazione

La sicurezza dei dati personali degli utenti è cruciale, ma garantirla a tutti i costi potrebbe limitare le opzioni di personalizzazione dell'avatar. Si cercherà un equilibrio tra sicurezza e possibilità di personalizzazione, mantenendo l'integrità dei dati senza compromettere eccessivamente la flessibilità nell'esperienza di prova dei vestiti sull'avatar.

Interattività VS Efficienza

Dato che l'obiettivo è fornire una simulazione interattiva dei vestiti sull'avatar, si porrà un'enfasi sull'interattività dell'applicazione, anche se ciò richiede un utilizzo più elevato delle risorse di sistema a beneficio dell'esperienza utente.

Componenti off-the-shelf (Pietro Lagioia, Guerino Pio Altieri)

Nella creazione del nostro software incorporeremo componenti off-the-shelf già disponibili per agevolare lo sviluppo del progetto.

Per lo sviluppo del lato front-end, utilizzeremo HTML5, CSS3 e JavaScript.

Per lo sviluppo del lato back-end, utilizzeremo come linguaggio di programmazione Python.

Ci serviremo del software MySQL per implementare il Database Relazionale.

Linee Guida per la documentazione delle interfacce (Tutti i membri del team)

Organizzazione del codice in file

È fondamentale organizzare il codice per la piattaforma DressMe Now in file distinti, ciascuno identificabile da un nome unico e sviluppato secondo la sua categoria. La struttura del sistema seguirà linee guida architetturali definite durante la progettazione, organizzando l'implementazione in pacchetti. Ogni nuovo documento avrà un nome intuitivo che riflette il suo scopo e sarà collocato nel package appropriato per una gestione agevole.

Linee guida per classi e interfaccia in Python

Per ciascun file di classe Python implementato, si dovranno rispettare i seguenti criteri:

- Classi e interfacce:
  - La nomenclatura delle classi deve seguire la convenzione UpperCamelCase;
  - Il nome delle classi deve essere chiaro e in linea con la loro funzione;
  - Il nome delle classi deve essere un singolo termine (es. Array) o al massimo un termine composto (es. ArrayList);
  - I nomi delle classi di test devono includere il nome della classe testata, seguito dal suffisso "TestUnit" nel caso di test unitari e "TestInteg" nel caso di test di integrazione;
- Costanti:
  - I valori immutabili definiti nelle classi Python devono essere dichiarati come "static final";
  - I nomi delle costanti devono essere scritti in maiuscolo;
  - La separazione tramite underscore è ammessa quando necessario;
- Variabili di istanza e variabili locali:
  - I nomi dei parametri o delle variabili locali devono seguire la convenzione lowerCamelCase o separati da underscore;

    - È necessario definire il livello di visibilità per ciascun parametro di istanza nella classe;
- Metodi:
    - I nomi dei metodi devono iniziare con un verbo;
    - È necessario specificare il livello di visibilità per ciascun metodo di una classe;
    - Eventuali parametri nella firma del metodo devono seguire le convenzioni adottate per le variabili di istanza;
- Blocchi e indentazioni:
    - Il codice deve essere indentato in modo accurato, utilizzando un Tab per ciascun livello di indentazione;
- Blocchi eccezionali:
    - Ogni blocco try-except definito all'interno di un metodo deve essere indentato correttamente secondo le specifiche sopra riportate;
    - Le operazioni comuni tra il blocco try e i blocchi except successivi devono essere posizionate nel blocco finally;
- Annotazioni:
    - Le annotazioni per una classe o un metodo devono apparire una per riga, subito dopo il blocco di documentazione;
    - Eventuali annotazioni prima di un parametro della classe devono essere dichiarate una per riga, sopra il parametro stesso;
- Commenti:
    - Ogni metodo della classe deve includere blocchi di commenti che chiariscano il flusso operativo del metodo;
    - È importante fornire commenti per spiegare operazioni complesse o blocchi di codice non immediatamente chiari;
    - La specifica dei commenti nel codice deve avvenire o ad ogni riga ponendo all'inizio della riga stessa un # oppure può avvenire in blocchi delimitati da tre virgolette: """.

Linee guida per pagine HTML 5

Per ogni file HTML5 creato, è necessario seguire i criteri seguenti:

- Ogni file deve includere il tag identificativo per la tecnologia HTML5 utilizzata;
- Ogni tag aperto nel corpo del documento HTML, ad eccezione dei tag singoli, richiede la presenza del rispettivo tag di chiusura;
- La struttura di base di una pagina HTML (head, body) deve essere rispettata;
- Ogni file HTML, soprattutto nel corpo del documento, deve essere opportunamente indentato per ciascuna definizione di un nuovo tag, preferibilmente utilizzando una tabulazione per ogni livello.

Linee guida per script JavaScript

- Ogni funzione Javascript dovrà essere riportata o in un documento diverso dalla pagina html inclusa, o nel documento stesso;

- I nomi di funzioni, variabili e costanti dovranno seguire le stesse specifiche definite per i documenti Python;
- Ogni script dovrà essere incluso successivamente al body del relativo documento HTML.

Linee guida per i Fogli di stile CSS3:

- Ogni set di istruzioni CSS non inline sarà posizionato in un file separato rispetto alla pagina HTML di riferimento, facilitando il riutilizzo evitando duplicazioni;
- Ove ciò non è possibile, ogni set di istruzioni CSS verrà posizionato tra l'head e il body;
- Ciascuna direttiva CSS inizierà su una nuova riga, con la dichiarazione dei selettori relativi alla direttiva stessa;
- L'ultimo selettore di una direttiva CSS sarà seguito dall'apertura della parentesi graffa {;
- L'indentazione dovrà seguire queste linee guida:
    - All'inizio di una nuova direttiva (#...{) e alla chiusura del blocco della direttiva (}), l'indentazione sarà a livello 0;
    - Le proprietà di ogni direttiva CSS saranno indentate da un Tab rispetto all'inizio del blocco e inserite su singole righe.

Design Pattern (Carlo Di Vico, Alfredo Baratta, Christian Iervasi)

Abbiamo adottato l'approccio con il design pattern Facade.

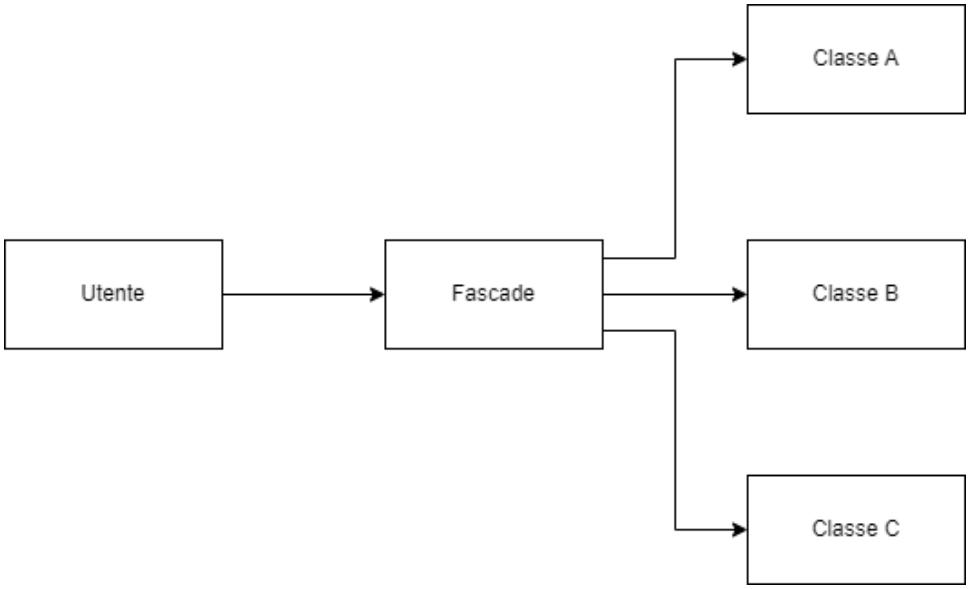
Problema

In molte applicazioni, specialmente quelle di grandi dimensioni, si può verificare la presenza di un elevato numero di classi correlate che forniscono funzionalità complesse. Questa complessità può risultare in un insieme intricato di interfacce, rendendo difficile per i client capire quale sia l'interfaccia essenziale da utilizzare per interagire con il sottosistema. La gestione delle comunicazioni e delle dipendenze dirette tra i client e i vari sottosistemi può diventare complicata e disorientante. È necessario semplificare l'accesso a tali funzionalità complesse riducendo la complessità dell'interfaccia e semplificando le interazioni tra client e sottosistema.

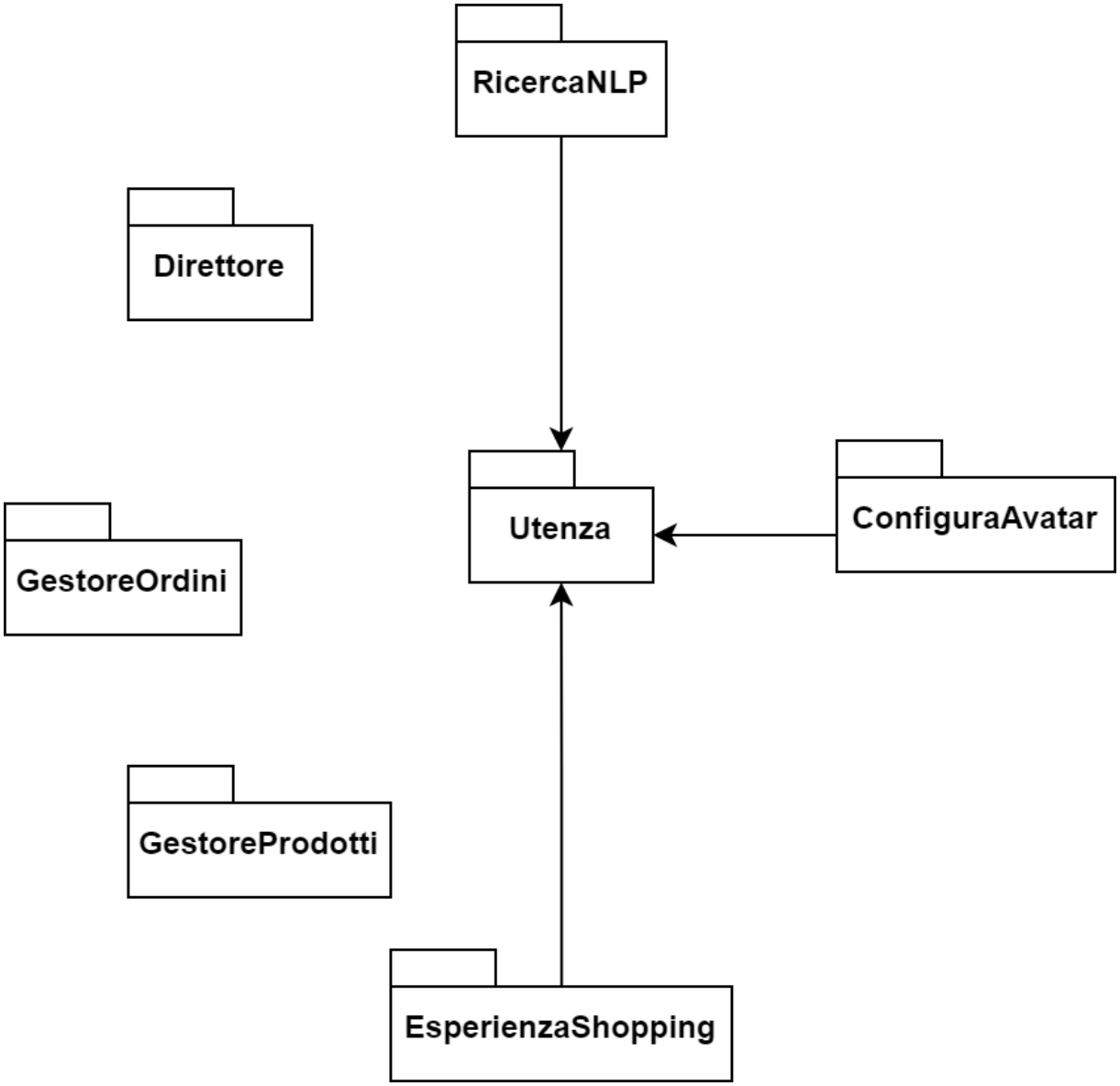
Soluzione

Il design pattern Facade offre una soluzione fornendo un'interfaccia unificata e semplificata ai client, nascondendo la complessità dei sottosistemi dietro un'unica interfaccia di alto livello. La sua funzione principale è quella di fornire un punto di accesso unificato per un insieme di funzionalità all'interno del sottosistema, riducendo così la complessità delle chiamate e semplificando l'interazione per i client. Il Facade agisce come intermediario tra il client e le classi o sottosistemi complessi, incapsulando la logica di accesso e gestione delle funzioni interne.
In pratica, il Facade si interpone tra il client e le molteplici classi o sottosistemi complessi, offrendo un'interfaccia semplificata che esegue chiamate ai metodi degli oggetti nascosti. In questo modo, il client interagisce solo con il Facade, senza dover preoccuparsi della complessità sottostante del sottosistema. Il Facade si occupa di dirigere le richieste del client verso le classi appropriate all'interno del sottosistema, nascondendo la sua complessità e semplificando così l'utilizzo del sistema nel suo complesso.

In sintesi, il design pattern Facade permette di rendere i sottosistemi più leggibili, facili da utilizzare e riduce la dipendenza dei client dalla complessità interna dei sottosistemi, fornendo un'interfaccia semplificata e unificata che rende più agevole l'interazione con il sistema complesso.


Classe A
Utente
Fascade
Classe B
Classe C

Packages (Alfredo Baratta, Christian Iervasi)



La divisione in package sovra rappresentata segue la divisione in sottosistemi definita precedentemente nella fase di system design, e che comprende:

- **Utenza:** Contiene tutti i file inerenti all'utente;
- **ConfiguraAvatar:** Contiene tutti i file inerenti alla configurazione dell'Avatar;
- **RicercaNLP:** Contiene tutti i file inerenti alla ricerca NLP;
- **EsperienzaShopping:** Contiene tutti i file inerenti all'esperienza di shopping;
- **Direttore:** Contiene tutti i file inerenti al Direttore;
- **GestoreOrdini:** Contiene tutti i file inerenti al gestore e alla gestione degli Ordini;
- **GestoreProdotti:** Contiene tutti i file inerenti al gestore e alla gestione dei Prodotti.

Package Utenza (Tutti i membri del team)

Classe	Descrizione
Utente.py	Classe che rappresenta l'entità Utente
Indirizzo.py	Classe che rappresenta l'entità Indirizzo
profilo.py	Controller per la gestione delle operazioni relative agli utenti
pagamenti.py	Controller per la gestione delle operazioni relative ai pagamenti
login.py	Controller per la gestione del login utente
Registrati.py	Controller per la gestione della registrazione utente
profilo.html	Pagina per la visualizzazione dei dati personali dell'utente
Ordini.html	Pagina che consente la visualizzazione degli ordini effettuati dall'utente
RicercaProdotti.html	Pagina relativa ai risultati di ricerca
Login.html	Pagina relativa al login dell'utente
Pagamento.html	Pagina relativa al pagamento
Indirizzi.html	Pagina per la visualizzazione, la modifica ed eliminazione degli indirizzi dell'utente
Aggiungi_indirizzo.html	Pagina l'aggiunta degli indirizzi dell'utente
Impostazioni.html	Pagina per il logout e la cancellazione dell'account utente
Ordini.html	Pagina per la visualizzazione degli ordini effettuati dall'utente
Pagamento.html	Pagina per l'inserimento dei dati di pagamento
registrazione.html	Pagina relativa alla registrazione di un nuovo utente

Package Direttore (Tutti i membri del team)

Classe	Descrizione
personale.py	Classe che rappresenta l'entità Direttore
Gestione_utenti.html	Controller per la gestione delle operazioni relative al direttore
AggiuntaPersonale.html	Pagina che consente l'aggiunta di personale
Login.html	Pagina che consente il login al Direttore
ModificaPersonale.html	Pagina che consente di modificare il personale attuale
Statistiche.html	Pagina che consente di visualizzare le statistiche del sito
Index.html	Pagina di index del Direttore
Modifica_utente.html	Pagina che consente di modificare i dati di un utente al Direttore

Package EsperienzaShopping (Tutti i membri del team)

Classe	Descrizione
Carrello.py	Classe che rappresenta l'entità Carrello
Taglia.py	Classe che rappresenta l'entità Taglia
Immagine.py	Classe che rappresenta l'entità Immagine
Ordine.py	Classe che rappresenta l'entità Ordine
Prodotto.py	Classe che rappresenta l'entità Prodotto
ProdottoInOrdine.py	Classe che rappresenta l'entità ProdottoInOrdine
carrello.py	Controller per la gestione delle operazioni relative al carrello
prodotti.py	Controller per la gestione delle operazioni relative ai prodotti
effettua_ordine.py	Controller per la gestione delle operazioni relative agli ordini
effettua_reso.py	Controller per la gestione delle operazioni relative ai resi
Transazione.py	Controller per la gestione delle operazioni relative alla transazione
Carrello.html	Pagina per la visualizzazione del carrello e dei prodotti al suo interno
Categoria.html	Pagina per la visualizzazione della categorie
DettaglioProdootto.html	Pagina per la visualizzazione dei dettagli di un prodotto selezionato
Index.html	homepage
resoOrdine.html	Pagina relativa al reso
RicercaProdotto.html	Pagina relativa ai risultati della ricerca
ConfermaOrdine.html	Pagina di conferma di effettuazione dell'ordine

Package ConfiguraAvatar (Tutti i membri del team)

Classe	Descrizione
ConfigurazioneAvatar.py	Classe che rappresenta l'entità ConfigurazioneAvatar
avatar.py	Controller per la gestione delle operazioni relative alla configurazione avatar
personalizzazioneAvatar.html	Pagina per la configurazione dell'avatar
provaSuAvatar.html	Pagina per la visualizzazione dei prodotti selezionati sull'avatar dell'utente

Package RicercaNLP (Tutti i membri del team)

Classe	Descrizione
ricercaNLP.py	Controller per la gestione delle operazioni relative alla ricerca dei prodotti tramite NLP
ricercaNLP.html	Pagina relativa alla ricerca di un prodotto tramite NLP
Risultati_NLP.html	Pagina relativa ai risultati della ricerca tramite NLP

Package GestoreOrdini (Tutti i membri del team)

Classe	Descrizione
Ordine.py	Classe che rappresenta l'entità GestoreOrdine
Login.html	Pagina che consente il login del gestore degli ordini
ModificaOrdine.html	Pagina che consente di modificare un ordine di un utente
GestoreOrdine.html	Pagina che consente di visualizzare gli ordini degli utenti

Package GestoreProdotti (Tutti i membri del team)

Classe	Descrizione
Prodotti.py	Controller per la gestione delle operazioni relative ai prodotti
Prodotto.py	Classe che rappresenta l'entità prodotto
Immagine.py	Controller per la gestione delle operazioni relative alle immagini dei prodotti
aggiungiImgProdotto.html	Pagina che consente di aggiungere un immagine ad un prodotto
aggiungiProdotto.html	Pagina che consente di aggiungere un prodotto
AggiungiTaglia.html	Pagina che consente di aggiungere taglia e quantità ad un prodotto
Login.html	Pagina che consente il login del gestore dei prodotti
ModificaProdotto.html	Pagina che consente di modificare un prodotto
gestoreProdotto.html	Pagina principale che consente la visualizzazione dei prodotti

Class Interfaces (Tutti i membri del team)

Nome File	**carrello.py**
Descrizione	Consente di gestire le richieste relative alle operazioni della sezione del carrello quali ad esempio visualizzazione, inserimento e rimozione dei prodotti.
Metodi	• aggiungi_al_carrello(req, res) • rimuovi_dal_carrello(req, res) • visualizza_carrello(req, res) • svuota_carrello(req, res)
Invarianti	-
Nome metodo	**+ aggiungi_al_carrello (req, res)**
Descrizione	Questo metodo consente all'utente di aggiungere un prodotto al carrello.
Pre-condizione	**context**: carrello::aggiungi_al_carrello(req, res) **pre:** req != null && res != null
Post-condizione	/
Nome metodo	**+rimuovi_dal_carrello(req, res)**
Descrizione	Questo metodo consente all'utente di rimuovere un prodotto dal carrello.
Pre-condizione	**context**: carrello::rimuovi_dal_carrello(req, res) **pre:** req != null && res != null
Post-condizione	/
Nome metodo	**+visualizza_carrello(req, res)**
Descrizione	Questo metodo consente all'utente di visualizzare i prodotti presenti nel carrello.
Pre-condizione	**context**: carrello::visualizza_carrelloreq, res) **pre:** req != null && res != null
Post-condizione	/
Nome metodo	**+svuota_carrello(req, res)**
Descrizione	Questo metodo consente all'utente di rimuovere tutti i prodotti dal carrello.
Pre-condizione	**context**: carrello::svuota_carrello(req, res)

	pre: req != null && res != null
Post-condizione	/

Nome File	prodotti.py
Descrizione	Consente di gestire tutte le operazioni possibili da parte di un cliente all'interno del sito relative ad alcune interazioni tra utente e prodotti, ad esempio visualizzazione dettagli di un prodotto, ricerca di un prodotto, visualizza prodotti,
Metodi	• search_products(req, res) • products_category(req, res) • view_product(req, res)
Invarianti	-
Nome metodo	**+search_products(req, res)**
Descrizione	Questo metodo consente all'utente di ricercare un prodotto tramite ricerca testuale all'interno del database.
Pre-condizione	**context**: prodotti::search_products(req, res) **pre:** req != null && res != null
Post-condizione	/
Nome metodo	**+products_category(req, res)**
Descrizione	Questo metodo consente all'utente di visualizzare i prodotti per categoria.
Pre-condizione	**context**: prodotti::products_category(req, res) **pre:** req != null && res != null
Post-condizione	/
Nome metodo	**+view_products(req, res)**
Descrizione	Questo metodo consente all'utente di visualizzare i dettagli di un prodotto.
Pre-condizione	**context**: prodotti::view_products(req, res) **pre:** req != null && res != null
Post-condizione	/

Nome File	pagamenti.py
Descrizione	Consente di gestire le richieste relative alle transazioni in fase di pagamento.
Metodi	• conferma_ordine(req, res) • pagamento(req, res) • verifica_pagamento(req, res)
Invarianti	-
Nome metodo	**+conferma_ordine(req, res)**
Descrizione	Questo metodo consente all'utente ricevere la conferma dell'ordine.
Pre-condizione	**context**: pagamenti::conferma_ordine(req, res) **pre:** req != null && res != null
Post-condizione	/
Nome metodo	**+pagamento(req, res)**
Descrizione	Questo metodo consente all'utente di arrivare alla pagina di inserimento dati di pagamento.
Pre-condizione	**context**: pagamenti::pagamento(req, res) **pre:** req != null && res != null
Post-condizione	/
Nome metodo	**+verifica_pagamento(req, res)**
Descrizione	Questo metodo consente la validazione del pagamento.
Pre-condizione	**context**: pagamenti::verifica_pagamento(req, res) **pre:** req != null && res != null
Post-condizione	/

Nome File	direttore.py
Descrizione	Consente di gestire le richieste, da parte del Direttore, relative all'aggiunta, la rimozione, la visualizzazione del personale; e la visualizzazione delle statistiche.
Metodi	• aggiungi_personale(req, res) • modifica_personale(req, res) • rimuovi_personale(req, res) • visualizza_statische_ordini(req, res) • modifica_utente(req, res) • mostra_utente(req, res) • mostra_presonale(req, res) • visualizza_utenti(req, res) • visualizza_personale(req, res)
Invarianti	-
Nome metodo	**+aggiungi_presonale(req, res)**
Descrizione	Questo metodo consente al Direttore di aggiungere personale.
Pre-condizione	**context**: direttore::aggiungi_personale(req, res) **pre:** req != null && res != null
Post-condizione	/
Nome metodo	**+ modifica_personale(req, res)**
Descrizione	Questo metodo consente al Direttore di modificare informazioni del personale.
Pre-condizione	**context**: direttore::modifica_personale(req, res) **pre:** req != null && res != null
Post-condizione	/
Nome metodo	**+rimuovi_personale(req, res)**
Descrizione	Questo metodo consente al Direttore di rimuovere personale.
Pre-condizione	**context**: direttore::rimuovi_personale(req, res) **pre:** req != null && res != null
Post-condizione	/
Nome metodo	**+visualizza_statistiche_ordini(req, res)**

Descrizione	Questo metodo consente al Direttore di visualizzare le statistiche.
Pre-condizione	**context**: direttore::visualizza_statistiche_ordini(req, res) **pre:** req != null && res != null
Post-condizione	/
Nome metodo	**+modifica_utente(req, res)**
Descrizione	Questo metodo consente al Direttore di modificare un utente.
Pre-condizione	**context**: direttore::modifica_utente (req, res) **pre**: req != null && res != null
Post-condizione	/
Nome metodo	**+mostra_utente(req, res)**
Descrizione	Questo metodo consente al Direttore di visualizzare un utente.
Pre-condizione	**context**: direttore::mostra_utente(req, res) **pre**: req != null && res != null
Post-condizione	/
Nome metodo	**+mostra_personale(req, res)**
Descrizione	Questo metodo consente al Direttore di visualizzare il personale.
Pre-condizione	**context**: direttore::mostra_personale(req, res) **pre**: req != null && res != null
Post-condizione	/
Nome metodo	**+visualizza_utenti (req, res)**
Descrizione	Questo metodo consente al Direttore di visualizzare tutti gli utenti.
Pre-condizione	**context**: direttore::visualizza_utenti(req, res) **pre**: req != null && res != null
Post-condizione	/
Nome metodo	**+visualizza_personale (req, res)**

Descrizione	Questo metodo consente al Direttore di visualizzare tutto il personale
Pre-condizione	**context**: direttore::visualizza_personale(req, res) **pre**: req != null && res != null
Post-condizione	/

Nome File	ordini.py
Descrizione	Consente di gestire le richieste, da parte del Gestore ordini, relative ad esempio alla modifica e la visualizzazione degli ordini relativi agli utenti
Metodi	• modifica_ordine(req, res) • visual_ordine(req, res) • dashboard(req, res) • cancella_ordine(req, res) • modifica_stato_ordine(req, res)
Invarianti	-
Nome metodo	**+ modifica_ordine(req, res)**
Descrizione	Questo metodo consente al Gestore Ordine di modificare un ordine.
Pre-condizione	**context**: ordini::modifica_ordine(req, res) **pre:** req != null && res != null
Post-condizione	/
Nome metodo	**+ visual_ordine(req, res)**
Descrizione	Questo metodo consente al Gestore Ordine di visualizzare un singolo ordine.
Pre-condizione	**context**: ordini::visual_ordine(req, res) **pre:** req != null && res != null
Post-condizione	/
Nome metodo	**+ dashboard(req, res)**
Descrizione	Questo metodo consente al Gestore Ordine di visualizzare tutti gli ordini in una dashboard.
Pre-condizione	**context**: ordini::dashboard (req, res) **pre**: req != null && res != null
Post-condizione	/
Nome metodo	**+ cancella_ordine (req, res)**
Descrizione	Questo metodo consente al Gestore Ordine di cancellare un ordine.
Pre-condizione	**context**: ordini::cancella_ordine(req, res)

	pre: req != null && res != null
Post-condizione	/
Nome metodo	**+ modifica_stato_ordine(req, res)**
Descrizione	Questo metodo consente al Gestore Ordine di modificare lo stato di un ordine.
Pre-condizione	**context**: ordini::modifica_stato_ordine (req, res) **pre**: req != null && res != null
Post-condizione	/

Nome File	prodotti.py
Descrizione	Consente di gestire le richieste, da parte del Gestore prodotti, relative ad esempio all'aggiunta di prodotti, alla modifica di un prodotto, alla rimozione di un prodotto, alla visualizzazione dei prodotti.
Metodi	• aggiungi_prodotto(req, res) • modifica_prodotto(req, res) • elimina_prodotto(req, res) • prodotti(req, res) • mostra_prodotto(req, res) • mostra_info_prodotto(req, res)
Invarianti	-
Nome metodo	**+ aggiungi_prodotto(req, res)**
Descrizione	Questo metodo consente al Gestore Prodotti di aggiungere un prodotto.
Pre-condizione	**context**: prodotti::aggiungi_prodotto(req, res) **pre:** req != null && res != null
Post-condizione	/
Nome metodo	**+ modifica_prodotto(req, res)**
Descrizione	Questo metodo consente al Gestore Prodotti di modificare un prodotto.
Pre-condizione	**context**: prodotti::modifica_prodotto(req, res) **pre:** req != null && res != null
Post-condizione	/
Nome metodo	**+ elimina_prodotto(req, res)**
Descrizione	Questo metodo consente al Gestore Prodotti di eliminare un prodotto
Pre-condizione	**context**: prodotti::elimina_prodotto(req, res) **pre:** req != null && res != null
Post-condizione	/
Nome metodo	**+prodotti (req, res)**
Descrizione	Questo metodo consente al Gestore Prodotti di visualizzare tutti i prodotti.

Pre-condizione	**context**: prodotti::prodotti (req, res) **pre:** req != null && res != null
Post-condizione	/
Nome metodo	**+ mostra__prodotto(req, res)**
Descrizione	Questo metodo consente al Gestore Prodotti di visualizzare un singolo prodotto.
Pre-condizione	**context**: prodotti::mostra_prodotto(req, res) **pre:** req != null && res != null
Post-condizione	/
Nome metodo	**+ mostra_info_prodotto(req, res)**
Descrizione	Questo metodo consente al Gestore Prodotti di visualizzare le informazioni di un prodotto.
Pre-condizione	**context**: prodotti::mostra_info_prodotto(req, res) **pre:** req != null && res != null
Post-condizione	/

Nome File	RicercaNLPController.py
Descrizione	Consente di ricercare un prodotto in base al proprio stile e alle preferenze personali, tramite il modulo di intelligenza artificiale
Metodi	• Ricerca(req, res)
Invarianti	-
Nome metodo	**+ Ricerca(req, res)**
Descrizione	Questo metodo consente all'utente di ricercare il prodotto mediante il modulo di AI.
Pre-condizione	**context**: RicercaNLPController::Ricerca(req, res) **pre:** req != null && res != null
Post-condizione	/

Nome File	avatar.py
Descrizione	Consente di utilizzare tutte le funzioni relative all'avatar personalizzabile, ad esempio, la personalizzazione dell'avatar o visualizzare un prodotto sull'avatar.
Metodi	• modifica_avatar(req, res) • prova_su_avatar (req, res)
Invarianti	-
Nome metodo	**+ modifica_avatar(req, res)**
Descrizione	Questo metodo consente all'utente di modificare le caratteristiche dell'avatar come l'altezza o la lunghezza dei capelli.
Pre-condizione	**context**: avatar::modifica_avatar(req, res) **pre:** req != null && res != null
Post-condizione	/
Nome metodo	**+ prova_su_avatar(req, res)**
Descrizione	Questo metodo consente all'utente di visualizzare un prodotto indossato dall'avatar personalizzabile.
Pre-condizione	**context**: avatar::prova_su_avatar(req, res) **pre:** req != null && res != null
Post-condizione	/

Nome file	login.py
Descrizione	Consente all'utente di effettuare il login
Metodi	• login(req, res)
Invarianti	-
Nome metodo	**+login(req, res)**
Descrizione	Questo metodo consente il login dell'utente
Pre-condizione	**Context:** login::login(req, res) **pre:** req != null && res != null
Post-condizione	/

Nome file	registrati.py
Descrizione	Consente all'utente di effettuare
Metodi	• register(req, res)
Invarianti	-
Nome metodo	**+register(req, res)**
Descrizione	Questo metodo consente ls registrazione dell'utente
Pre-condizione	**Context:** registrati::register(req, res) **pre:** req != null && res != null
Post-condizione	/

Class Diagram (Tutti i membri del team)