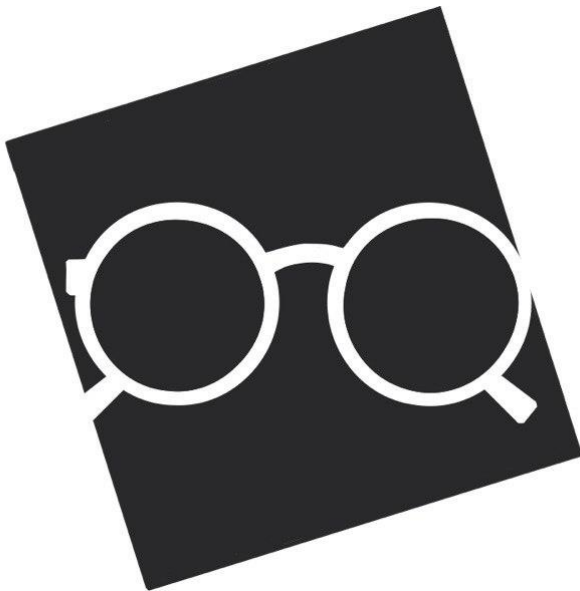


Università degli Studi di Salerno
Corso di Ingegneria del Software

The Spectacles
Object Design Document
Versione 0.4



The
Spectacles

Data: 13/01/2023

Progetto: The Spectacles	Versione: 0.3
Documento: Object Design Document	Data: 13/01/2023

Partecipanti:

Nome	Matricola
Roberto Piscopo	0512109906
Alessandro Satta	0512110929
Mario Ranieri	0512110017
Luca Di Meglio	0512110767

Scritto da:	Alessandro Satta, Roberto Piscopo, Mario Ranieri, Luca Di Meglio
--------------------	--

Revision History

Data	Versione	Descrizione	Autore
13/01/2023	0.1	Inizio documentazione del progetto, inserimento dei dati relativi al punto 1	Luca Di Meglio
20/01/2023	0.2	Inserimento Tabelle UML entity, manager e Control	Roberto Piscopo
21/01/2023	0.3	Aggiunta Decomposizione in pacchetti	Alessandro Satta
12/02/2023	0.4	Modifiche varie alle tabelle UML	Mario Ranieri
14/02/2023	0.5	Modifica indice	Alessandro Satta

Indice

1.	INTRODUZIONE.....	
1.1	Object design Trade-off.....	
1.1.1	Tempo di rilascio VS Funzionalità	
1.1.2	Affidabilità VS Tempo di risposta.....	
1.1.3	Portabilità VS Efficienza.....	
1.2	Componenti off-the-shelf	
1.3	Linee guida per la documentazione dell'interfaccia	
1.3.1	Java.....	
1.3.2	Pagine HTML.....	
1.3.3	Pagine lato Server(JSP).....	
1.3.4	Script JavaScript.....	
1.3.5	Fogli di stile(CSS).....	
1.3.6	Database SQL.....	
1.4	Definizioni, acronimi e abbreviazioni	
1.5	Riferimenti	
2.	Packages	
2.1	Divisione in pacchetti.....	
2.2	Organizzazione del codice in file	
3.	Interfacce delle classi	
3.1	Entity	
3.2	Manager	
3.3	Control.....	
3.4	Extra	
4.	Glossario	

1. INTRODUZIONE

1.1.Object Design Trade-off

Nella fase di Trade-off bisogna scegliere quali caratteristiche rispettare e quali rendere opzionali, in quanto all'aumentare di una caratteristica diminuisce l'altra.

1.1.1 Tempo di rilascio VS Funzionalità

Per poter rilasciare un e-commerce quanto più privo di bug e con tutte le funzionalità necessarie al fine di avere un impatto positivo con i primi clienti si è preferito mettere in secondo piano il tempo di rilascio.

1.1.2 Affidabilità vs Tempo di risposta

L'affidabilità verrà preferita al tempo di risposta per poter garantire un controllo accurato dei dati in input per poter minimizzare gli errori.

1.1.3 Portabilità vs efficienza

Abbiamo garantito la portabilità utilizzando un linguaggio non nativo, Java è infatti indipendente dalla piattaforma anche se questa sua caratteristica comporta delle prestazioni inferiori rispetto ad altri linguaggi.

1.2 Componenti off-the-shelf

Per realizzare il The Spectacles verranno utilizzati componenti software già disponibili per semplificare la creazione del sistema.

The Spectacles, per quanto riguarda il front-end, utilizzerà la libreria JQuery.

JQuery è una libreria JavaScript che consentirà di scrivere parte della logica front-end dell'applicazione in maniera più veloce ed efficace e che aiuterà nell'implementazione della tecnologia AJAX.

Lato back-end saranno invece utilizzate le seguenti componenti off-the-shelf: MySQL (v.8.0), Tomcat (v.9.X).

MySQL è un sistema open source di gestione di database relazionali SQL sviluppato e supportato da Oracle.

Verrà utilizzato per l'implementazione del Database di The Spectacles.

Per la comunicazione con il database tramite Java verrà utilizzato la componente MySQL Connector/J, il driver JDBC ufficiale per la comunicazione con MySQL tramite Java.

Tomcat è un server web open-source su cui verrà eseguito tutto il back-end dell'applicazione.

Ci permetterà di utilizzare le Java Servlet per l'implementazione della logica di business e le Java Servlet Pages per la generazione dinamica delle pagine web utilizzate dal client.

1.3 Linee guida per la documentazione dell'interfaccia

Package: il nome del pacchetto deve essere sempre in minuscolo

Classi: i nomi delle classi devono essere descrittivi e scritti in UpperCamelCase

Interfacce: i nomi delle interfacce devono essere scritte in UpperCamelCase

Metodi: i metodi devono essere scritti in forma camelCase

Variabili: il nome delle variabili deve essere descrittivo, si evita quindi di utilizzare variabili con nomi composti da una sola lettera se non per variabili temporanee.

Il formato del nome delle variabili è il lowerCamelCase.

1.3.1 Java

Nella codifica di classi e interfacce Java, è opportuno rispettare le seguenti regole di formattazione:

1. Non inserire spazi tra il nome del metodo e la parentesi tonda "(" che apre la lista dei parametri.
2. La parentesi graffa aperta "{" si trova alla fine della stessa linea dell'istruzione di dichiarazione.
3. La parentesi graffa chiusa "}" inizia su una nuova riga vuota allo stesso livello di indentazione del nome della classe o dell'interfaccia.

Nel caso di istruzioni semplici, ogni linea deve contenere al massimo una sola istruzione, mentre nel caso di istruzioni composte vanno rispettate le seguenti regole:

1. Le istruzioni racchiuse all'interno di un blocco (esempio: for), devono essere indentate di un'unità all'interno dell'istruzione composta.
2. La parentesi di apertura del blocco deve trovarsi alla fine della riga dell'istruzione composta.
3. La parentesi di chiusura del blocco deve trovarsi allo stesso livello di indentazione dell'istruzione composta
4. Le istruzioni composte formate da un'unica istruzione devono essere racchiuse da parentesi.

1.3.2 Pagine HTML

Le pagine HTML, sia in forma statica che dinamica, devono essere conformi allo standard HTML 5. Inoltre, il codice HTML statico deve utilizzare l'indentazione, per facilitare la lettura, secondo le seguenti regole:

1. Un'indentazione consiste in una tabulazione;
2. Ogni tag deve avere un'indentazione maggiore del tag che lo contiene;
3. Ogni tag di chiusura deve avere lo stesso livello di indentazione del corrispondente tag di apertura;
4. I tag di commento devono seguire le stesse regole che si applicano ai tag normali.

1.3.3 Pagine lato Server (JSP)

Anche le pagine JSP, quando eseguite, devono produrre un documento conforme allo standard HTML 5. Il codice Java delle pagine deve aderire alle convenzioni per la codifica in Java, con i seguenti accorgimenti:

1. Il tag di apertura (<%) si trova all'inizio di una riga;
2. Il tag di chiusura (%>) si trova all'inizio di una nuova riga;
3. È possibile evitare le due regole precedenti, se il corpo del codice Java consiste in una singola istruzione.

1.3.4 Script JavaScript

Gli Script in JavaScript devono rispettare le seguenti convenzioni:

1. Gli script che svolgono funzioni differenti dal rendering della pagina dovrebbero essere collocati in file appositi.
2. Il codice JavaScript deve seguire le stesse convenzioni per il layout e i nomi del codice Java.

1.3.5 Fogli di stile (CSS)

I fogli di stile devono seguire le seguenti convenzioni:

Tutti gli stili non in-line devono essere collocati in fogli di stile separati.

Ogni regola CSS deve essere formattata come segue:

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.

1.3.6 Database SQL

I nomi delle tabelle devono seguire le seguenti regole:

1. Devono essere costituiti da sole lettere minuscole;
2. Se il nome è costituito da più parole, è previsto l'uso di underscore (_).

I nomi dei campi devono seguire le seguenti regole:

1. Devono essere costituiti da sole lettere minuscole;
2. Se il nome è costituito da più parole, è previsto l'uso di underscore (_);

1.4 Definizioni, acronimi e abbreviazioni

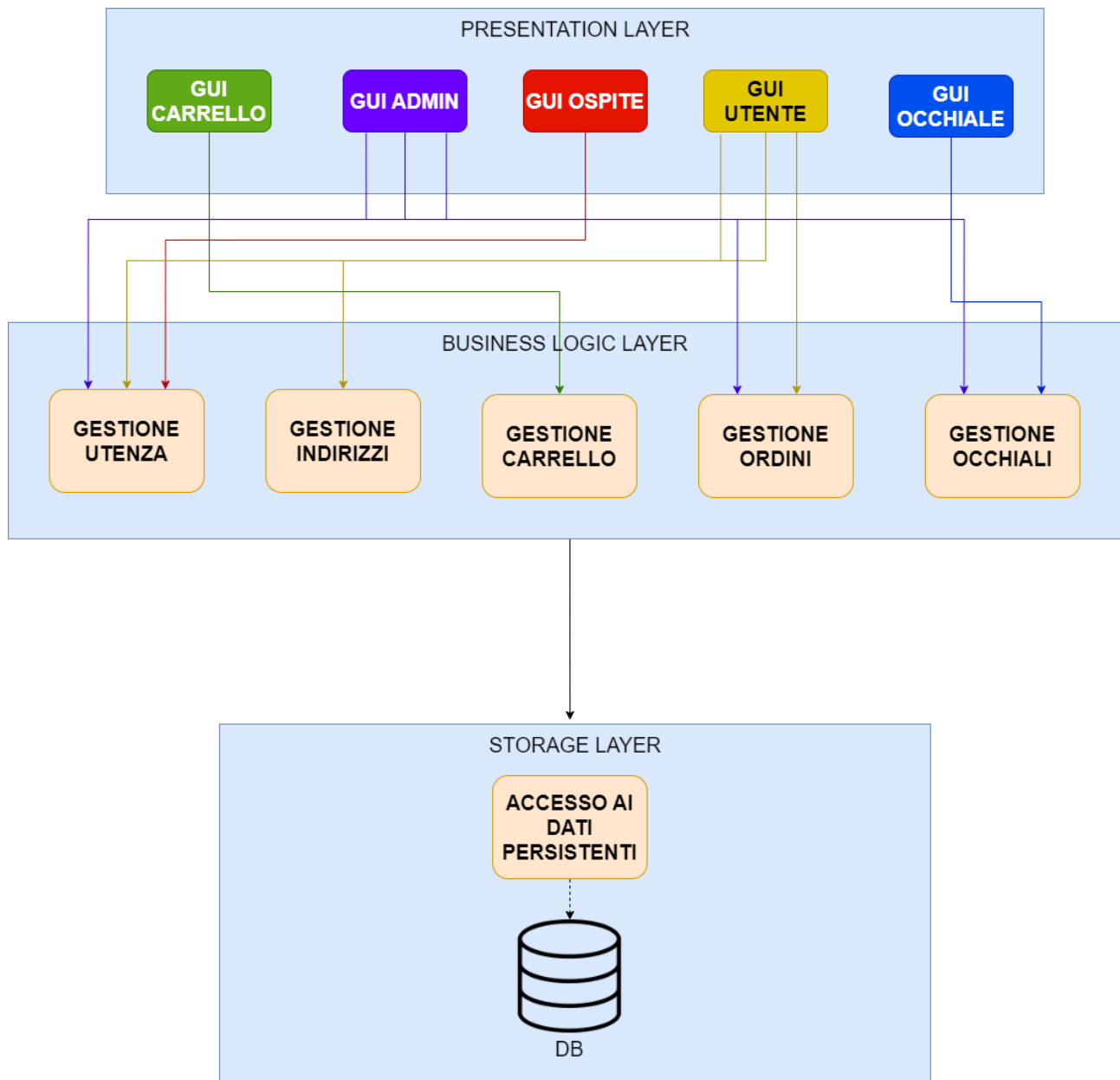
- **RAD:** Requirements Analysis Document.
- **SDD:** System Design Document.
- **ODD:** Object Design Document.
- **UC:** Use Case.
- **RF:** Requisito funzionale.
- **NRF:** Requisito non funzionale.
- **CSS:** acronimo di Cascading Style Sheets, è un linguaggio usato per definire la formattazione delle pagine Web.
- **JavaScript:** linguaggio di scripting orientato agli oggetti e agli eventi, comunemente utilizzato nella programmazione Web lato client per la creazione di effetti dinamici interattivi.
- **JQuery:** JQuery è una libreria JavaScript per applicazioni web.
- **AJAX:** acronimo di Asynchronous JavaScript and XML, è una tecnica di sviluppo software per la realizzazione di applicazioni web interattive.
- **lowerCamelCase:** tecnica di naming delle variabili, adottata dallo standard Google Java, che consiste nello scrivere più parole insieme delimitando la fine e l'inizio di una nuova parola con una lettera maiuscola.
- **Servlet:** oggetti scritti in linguaggio Java che operano all'interno di un server web.
- **Tomcat:** un web server open source. Implementa le specifiche JavaServer Pages (JSP) e servlet, fornendo quindi una piattaforma software per l'esecuzione di applicazioni Web sviluppate in linguaggio Java.

1.5 Riferimenti

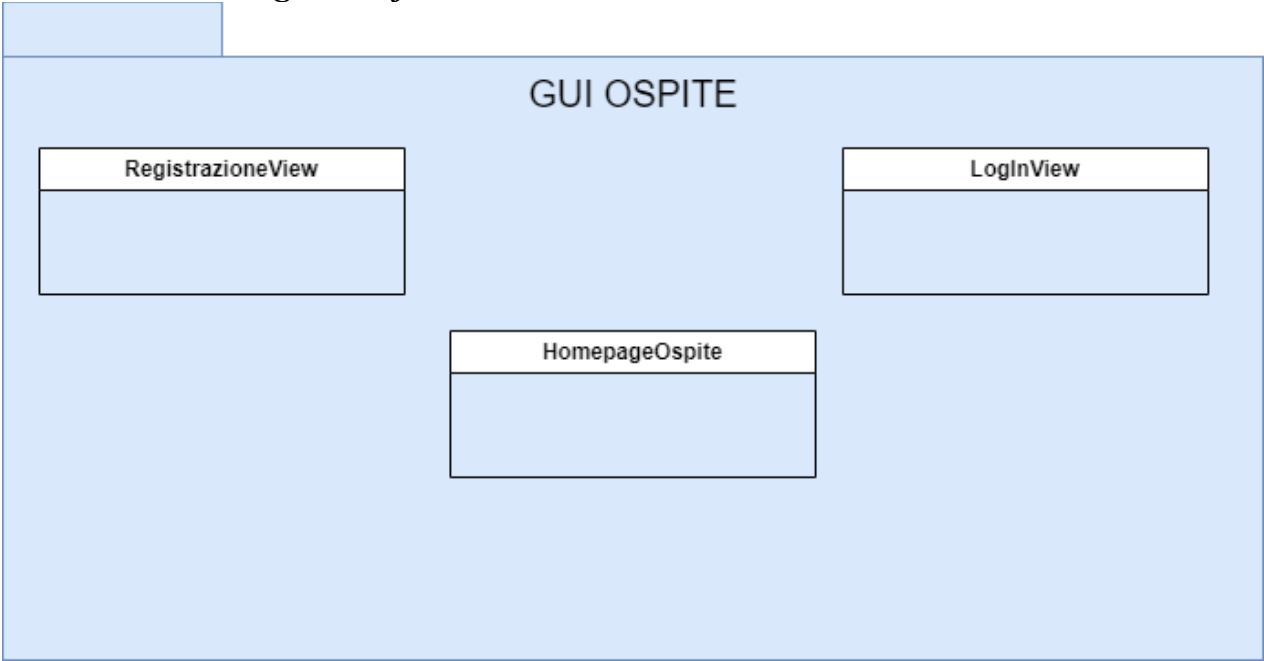
Object-Oriented Software Engineering Using UML, Patterns, and Java™ Third Edition Bernd Bruegge & Allen H. Dutoit
RAD_THE_SPECTACLES
SDD_THE_SPECTACLES

2. Packages

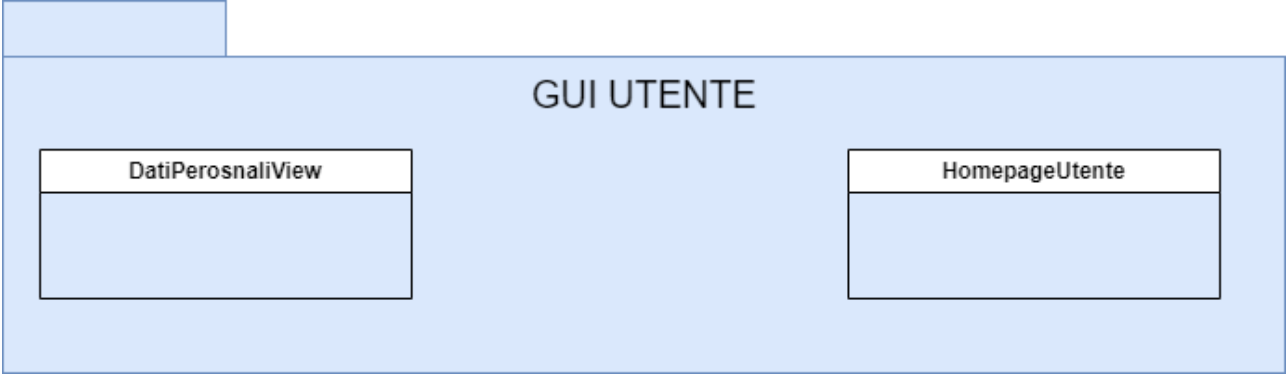
2.1 Divisione In Pacchetti



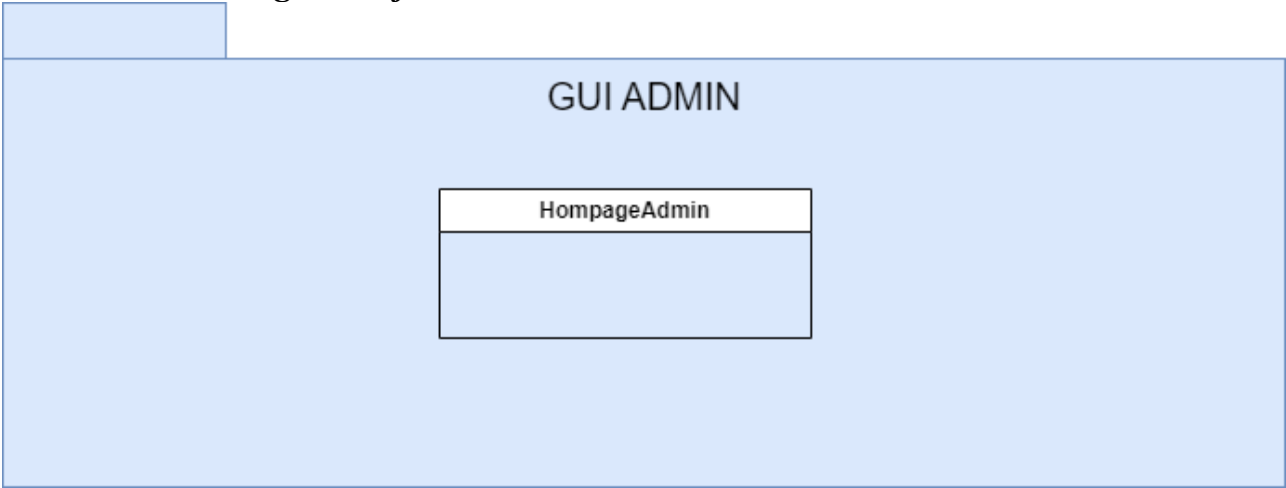
2.1.1 Package Interface Gestione Utente



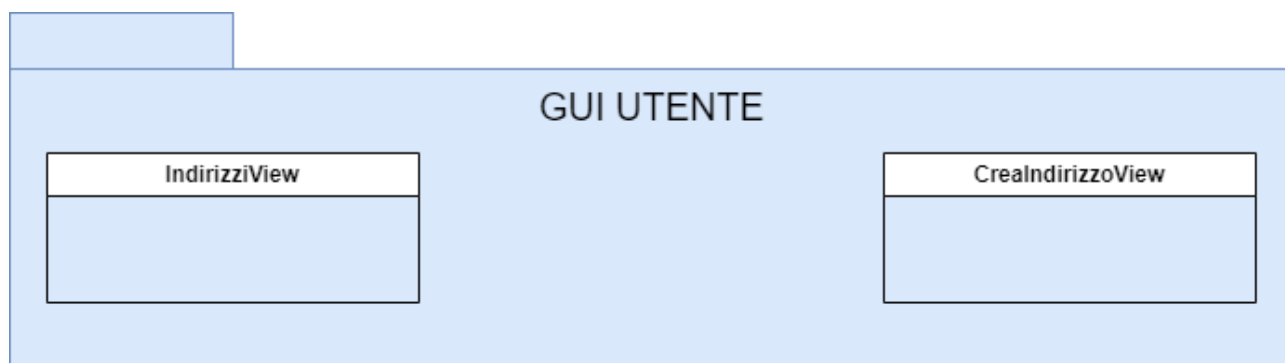
2.1.2 Package Interface Gestione Utente



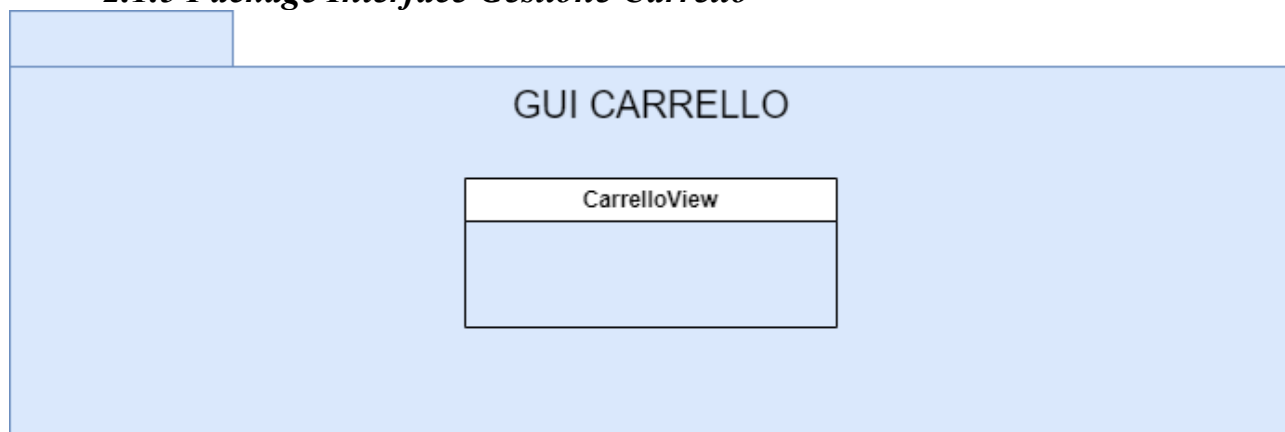
2.1.3 Package Interface Gestione Utente



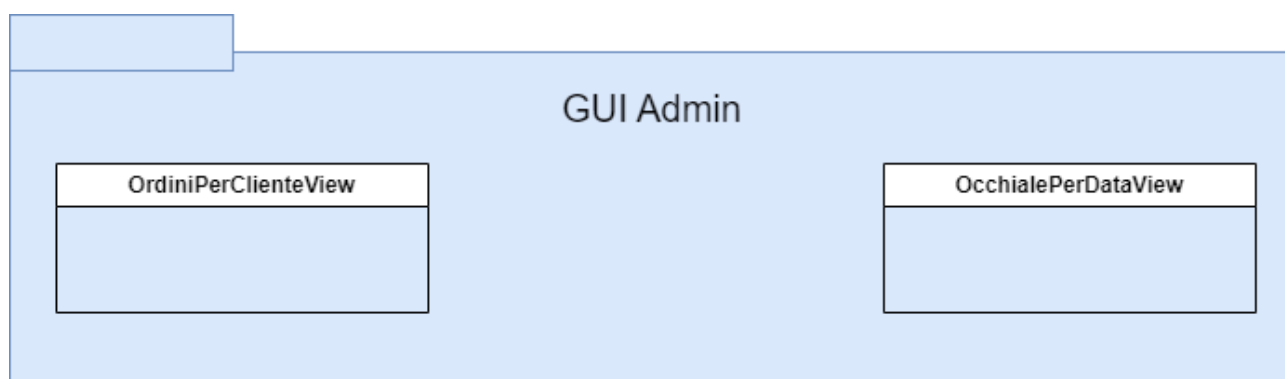
2.1.4 Package Interface Gestione Indirizzo



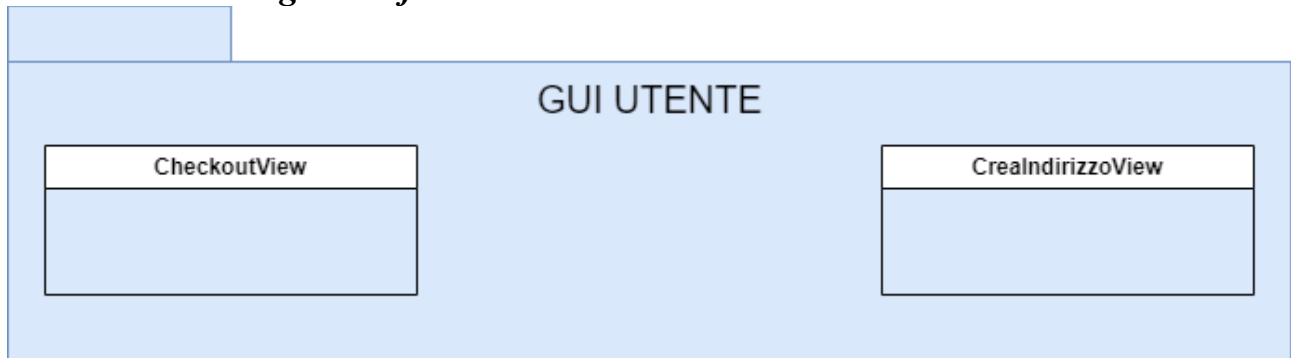
2.1.5 Package Interface Gestione Carrello



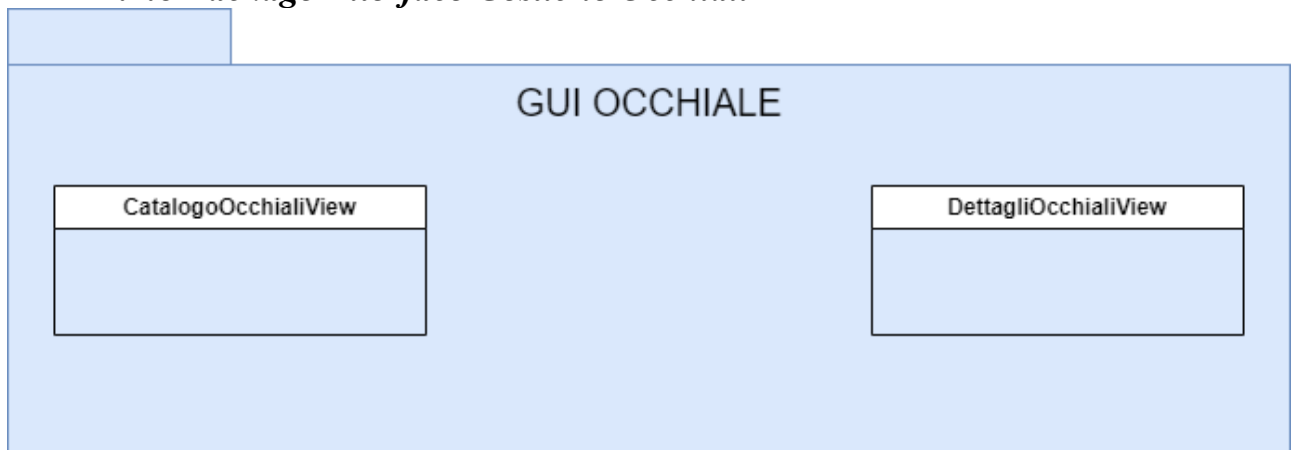
2.1.6 Package Interface Gestione Ordini



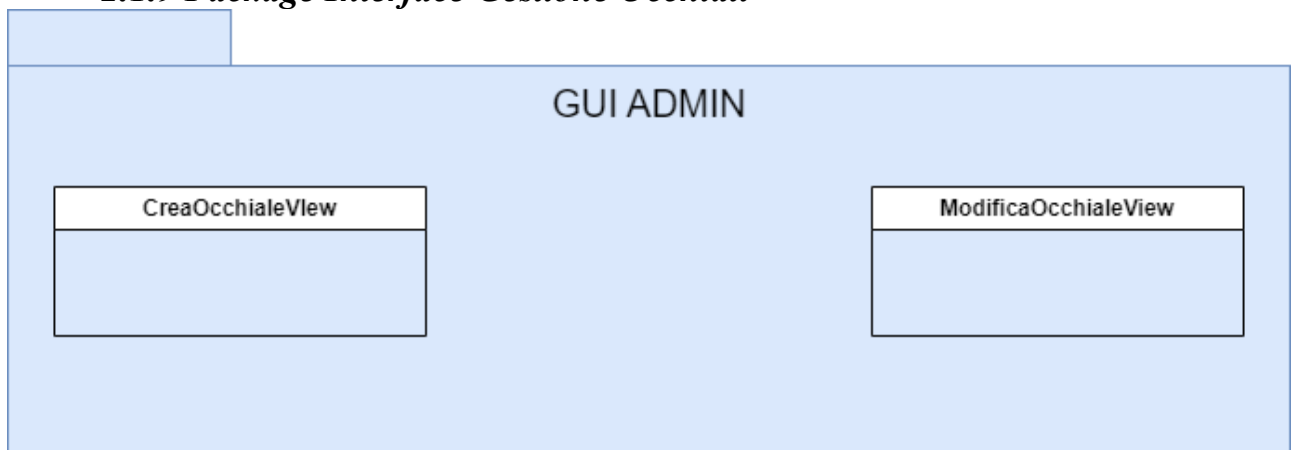
2.1.7 Package Interface Gestione Ordini



2.1.8 Package Interface Gestione Occhiali



2.1.9 Package Interface Gestione Occhiali



2.2 Organizzazione del codice in file

UtenteBean - pass: String - role: int - email String - firstName: String - lastName: String - birthday: Date	OcchialeBean - idGlasses: String - nameGlasses: String - brand: String - price: int - availability: int - type: String - color: String - category: String - image: String - image2: String - description: String - quantity: int - totalPrice: float	IndirizzoBean - idIndirizzo: int - Name: String - Surname: String - address: String - status: int - city: String - province: String - cap: int - email: String - telefono: int
OcchialeOrdineBean - idOcchialeOrdine: int - idOrdine: UUID - occhiale: OcchialeBean - idOcchiale: String - prezzoEffettivo: int - iva: float - quantita: int	Carrello - car: ArrayList<OcchialeBean> - dimensione: int - tot: float	OrdineBean - idOrder: UUID - data: Date - email: String - stato: String - tot: int

3. Interfacce delle calssi

3.1 Entity

Nome Classe	UtenteBean
Descrizione	Rappresenta l'oggetto utente.
Firma dei Metodi	+ getPass():String + getRole():void + getEmail():String + getFirstName():String + getLastName():String + getBirthday():Date +setPass(pass:String) +setRole(role:int) +setEmail(email:String) +setFirstName(firstName:String) +setLastName(String:lastName) +setBirthday(birthday:Date)
Pre-condizioni	Context UtenteBean::setRole(role) Pre: role==0 OR role==1 Context UtenteBean::setEmail(email) Pre: l'email non deve essere già presente nel DB
Post-condizioni	Context UtenteBean::setRole(role)

	Post: l'utente è standard o admin Context UtenteBean::setEmail(email) Post: l'email è nel DB
Invariante	Context UtenteBean Inv: role==0 OR role==1

Nome Classe	OcchialeBean
Descrizione	Rappresenta l'oggetto occhiale.
Firma dei Metodi	+getIdGlasses():String +setIdGlasses(idGlasses:String) +getNameGlasses():String +setNameGlasses(nameGlasses:String) +getBrand():String +setBrand(brand:String) +getPrice():int +setPrice(price:int) +getAvailability():int +setAvailability(availability:int) +getType():String +setType(type:String) +getColor():String +setColor(color:String) +getCategory():String +setCategory(category:String) +getImage():String +setImage(image:String) +getImage2():String +setImage2(image2:String) +getDescription():String +setDescription(description:String) +getQuantity():int +setQuantity(quantity:int) +getTotPrezzo():float +setTotPrezzo(prezzotot:float)
Pre-condizioni	Context OcchialeBean::setIdGlasses(idOcchiale) Pre: idOcchiale non deve essere già presente nel DB Context OcchialeBean::setQuantity(quantity) Pre: quantity non deve essere negativa
Post-condizioni	Context OcchialeBean::setIdGlasses(idOcchiale) Post: l'idOcchiale è presente nel DB
Invariante	Context OcchialeBean Inv: self.quantity>=0

--	--

Nome Classe	OrdineBean
Descrizione	Rappresenta l'oggetto ordine.
Firma dei Metodi	+getIdOrder():UUID +setIdOrder(idOrder:UUID) +getDate():Date +getStato():String +setStato(s:String) +setDate(date:Date) +getEmail():String +getTot():int +setTot(t:int) +setEmail(email:String)
Pre-condizioni	Context OrdineBean::setStato(stato) Pre: stato=="confermato" OR stato=="in elaborazione" OR stato=="spedito" Context OrdineBean::setIdOrder(idOrder) Pre: idOrder non deve essere già presente in DB Context OrdineBean::setEmail(email:String) Pre: email deve avere una corrispondenza nel DB come chiave di un UtenteBean
Post-condizioni	Context OrdineBean::setIdOrder(idOrder) Post: idOrder è presente in DB Context OrdineBean::setEmail(email:String) Post: email corrisponde ad un utente presente nel DB
Invariante	Context OrdineBean Inv: self.stato=="confermato" or self.stato=="in elaborazione" or self.stato=="spedito"

Nome Classe	OcchialeOrdineBean
Descrizione	Rappresenta i vari occhiali presenti in un Ordine.

Firma dei Metodi	<pre> +getIdOcchialeOrdine():int +getIdOrdine():UUID +getProdotto():OcchialeBean +getIdProdotto():String +getPrezzoEffettivo():float +getIva():float +getQuantita():int +setIdOcchialeOrdine(idOcchialeOrdine:int) +setIdOrdine(idOrdine:UUID) +setProdotto(occhiale:OcchialeBean) + setIdProdotto(idOcchiale:String) +setPrezzoEffettivo(prezzoEffettivo:int) +setQuantita(quantita:int) </pre>
Pre-condizioni	<p>Context OcchialeOrdineBean::setIdOcchialeOrdine(idOcchialeOrdine)</p> <p>Pre: idOcchialeOrdine non deve essere già presente nel DB</p> <p>Context OcchialeOrdineBean::setIdProdotto(idOcchiale)</p> <p>Pre: idOcchiale deve avere una corrispondenza nel DB come chiave di un OcchialeBean</p> <p>Context OcchialeOrdineBean::setIdOrdine(idOrdine)</p> <p>Pre: idOrdine deve avere una corrispondenza nel DB come chiave di un OrdineBean</p>
Post-condizioni	<p>Context OcchialeOrdineBean::setIdOcchialeOrdine(idOcchialeOrdine)</p> <p>Post: idOcchialeOrdine presente nel DB</p> <p>Context OcchialeOrdineBean::setIdProdotto(idOcchiale)</p> <p>Post: idOcchiale ha una corrispondenza nel DB come chiave di un OcchialeBean</p> <p>Context OcchialeOrdineBean::setIdOrdine(idOrdine)</p> <p>Post: idOrdine ha una corrispondenza nel DB come chiave di un OrdineBean</p>
Invariante	<p>Context OcchialeOrdineBean</p> <p>Inv: self.prezzoEffettivo>=0</p>

Nome Classe	IndirizziBean
Descrizione	Rappresenta l'oggetto indirizzo.

Firma dei Metodi	+getNome():String +setNome(nome:String) +getCognome():String +setCognome(cognome:String) +getIdIndirizzo():int +setIdIndirizzo(idIndirizzo:int) +getAddress():String +setAddress(address:String) +getStatus():int +setStatus(status:int) +getCity():String +setCity(city:String) +getProvince():String +setProvince(province:String) +getCap():int +setCap(cap:int) +getEmail():String +setEmail(email:String) +getTelefono():String +setTelefono(telefono:String)
Pre-condizioni	Context IndirizzoBean::setIdIndirizzo(idIndirizzo) Pre: idIndirizzo non è già presente nel DB Context IndirizzoBean::setEmail(email) Pre: email deve avere una corrispondenza nel DB come chiave di un UtenteBean
Post-condizioni	Context IndirizzoBean::setIdIndirizzo(idIndirizzo) Post: idIndirizzo è presente nel DB Context IndirizzoBean::setEmail(email) Post: email ha una corrispondenza nel DB come chiave di un UtenteBean
Invariante	Context IndirizzoBean Inv: self.Status==1 OR self Status==1

Nome Classe	Carrello
Descrizione	Rappresenta l'oggetto carrello.

Firma dei Metodi	+addCarrello(prod:OcchialeBean) +deleteProduct(prod:OcchialeBean) +prendiProdotto(id:String):OcchialeBean +delete() +getCarrello():ArrayList<OcchialeBean> +searchProdotto(code:String):boolean +getDimensione():int +getPrezzoTotale(quant:int,id:String):float +calcolaPrezzoTotale(id:String):float +modificaQuantita(id:String,quant:int) +setTotale(t:float) +getTotale():float
Pre-condizioni	Context Carrello::prendiProdotto(idOcchiale) Pre: idOcchiale deve essere presente nel DB Context Carrello::modificaQuantità(idOcchiale,quantità) Pre: idOcchiale deve essere presente nel DB e quantità deve essere >=0
Post-condizioni	Context Carrello::prendiProdotto(idOcchiale) Post: Un OcchialeBean viene utilizzato
Invariante	Context Carrello Inv: self.dimensioneCarrello>0

3.2 Manager

Nome Classe	UtenteDao
Descrizione	Rappresenta il manager che si occupa di interagire con il DB gestendo le query.
Firma dei Metodi	+doRetrieveByKey(keys:ArrayList<String>):UtenteBean +esisteEmail(email:String):boolean +doRetrieveByMail(email:String):UtenteBean +cercaIndirizzo(email:String):IndirizziBean +doRetrieveAll(order:String):Collection<UtenteBean> +changePassword(email:String, pass:String) +doDelete(bean:UtenteBean) +doSave(utente:UtenteBean)
Pre-condizioni	Context UtenteDao::esisteEmail(email) Pre: email!=null Context UtenteDao::doRetrieveByMail(email) Pre: email!=null
Post-condizioni	Context UtenteDao::esisteEmail(email)

	Post: true if db.utente->includes(select(u utente.email=email)), false altrimenti Context UtenteDao::doRetrieveByMail(email) Post: user=db.utente->(select(u utente.email=email))
--	--

Nome Classe	OcchialeDao
Descrizione	Rappresenta il manager che si occupa di interagire con il DB gestendo le query.
Firma dei Metodi	+doRetrieveByKey(keys:ArrayList<String>):Collection<OcchialeBean> +doRetrieveBySex(sex:String):Collection<OcchialeBean> +doRetrieveOcchiale (id:String): OcchialeBean +singleProduct(keys:ArrayList<String>):OcchialeBean +doRetrieveAll():Collection<OcchialeBean> +doRetrieveByBrand(brand:String):Collection<OcchialeBean> +decreaseAvailability(occhiale: OcchialeBean):void +doSave(occhiale:OcchialeBean):void +doDelete(occhiale:OcchialeBean):void +doUpdate(occhiale:OcchialeBean):void
Pre-condizioni	Context OcchialeDao:: doRetrieveOcchiale(id) Pre: id!=null Context OcchialeDao:: doRetrieveByBrand(brand) Pre: brand!=null
Post-condizioni	Context OcchialeDao::doRetrieveOcchiale(id) Post: occhiale=db.occhiale->(select(o o.idOcchiale=id)) Context OcchialeDao::doRetrieveByBrand(brand) Post: occhiale=db.occhiale->(select(o o.brand =brand))

Nome Classe	IndirizziDao
Descrizione	Rappresenta il manager che si occupa di interagire con il DB gestendo le query.
Firma dei Metodi	+doRetrieveActive(email: String): IndirizziBean +search(email: String, via: String): IndirizziBean +doRetrieveAllAddress(email: String): Collection<IndirizziBean> +doRetrieveAll(order: String): Collection<IndirizziBean> +doUpdate(indirizzo: IndirizziBean): void +doDelete(indirizzo: IndirizziBean): void +doSave(indirizzo: IndirizziBean):void
Pre-condizioni	Context IndirizziDao::doRetrieveActive(email)

	Pre: email!=null
Post-condizioni	Context IndirizziDao::doRetrieveActive(email) Post: indirizzo=db.indirizzi->(select(i i.email=email AND i.attivo=1))

Nome Classe	OrdineDao
Descrizione	Rappresenta il manager che si occupa di interagire con il DB gestendo le query.
Firma dei Metodi	+doRetrieveByKey(idOrdine:String):OrdineBean +doRetrieveByUser(email:String):ArrayList<OrdineBean> +doRetrieveAll(order:String):Collection<OrdineBean> +doRetriveByDate(init:Date, end:Date, skip:int, limit:int):ArrayList<OrdineBean> +doSave(ordine:OrdineBean):void +doUpdate(ordine:OrdineBean):void +doDelete(ordine:OrdineBean):void
Pre-condizioni	Context OrdineDao::doRetrieveAll(order) Pre: order!=null Context OrdineDao::doRetriveByUser(user) Pre: user!=null
Post-condizioni	Context OrdineDao::doRetrieveAll(order) Post: ordini=db.ordine->(select(o),orderBy order) Context OrdineDao::doRetriveByUser(user) Post: ordini=db.ordine->(select(o o.email=user))

Nome Classe	OcchialeOrdineDao
Descrizione	Rappresenta il manager che si occupa di interagire con il DB gestendo le query.
Firma dei Metodi	+doSave(occhialeOrdine:OcchialeOrdineBean):void +doDelete(idOcchialeOrdine:String):boolean +doRetrieveByKey(idOcchialeOrdine:String):OcchialeOrdineBean +doRetrieveAll(order:String):Collection<OcchialeOrdineBean> +doRetrivebyOrder(ordine:String,data:DataSource):ArrayList<OcchialeOrdineBean>
Pre-condizioni	Context OcchialeOrdineDao::doRetrieveByKey(idOcchialeOrdine) Pre: idOcchiale!=null Context OcchialeOrdineDao::doRetrivebyOrder(ordine,data) Pre: ordine!=null, data!=null

Post-condizioni	Context OcchialeOrdineDao::doRetrieveByKey(idOcchialeOrdine) Post: occhialeOrdine=db.occhialeOrdine->(select(o o.id=idOcchialeOrdine)) Context OcchialeOrdineDao::doRetrivebyOrder(ordine,data) Pre: ordini =db.occhialeOrdine->(select(o o.id_ordine=ordine))
-----------------	---

3.3 Control

Nome Classe	LoginServlet
Descrizione	Questa classe è un control che si occupa di passare a UtenteDao i dati dell'utente usando la query di Retrieve.
Firma dei Metodi	# doPost(request: HttpServletRequest, response: HttpServletResponse):void # doGet(request: HttpServletRequest, response: HttpServletResponse):void
Pre-condizioni	Context LoginServlet::doPost(request,response) Pre: request.getParameter("email")!=null request.getParameter("password")!=null
Post-condizioni	Context LoginServlet::doPost(request,response) Post: request.getSession().getAttribute("auth")!=null

Nome Classe	SigninServlet
Descrizione	Questa classe è un control che si occupa di passare a UtenteDao i dati dell'utente per inserire il nuovo utente.
Firma dei Metodi	# doPost(request: HttpServletRequest, response: HttpServletResponse):void # doGet(request: HttpServletRequest, response: HttpServletResponse):void
Pre-condizioni	Context SigninServlet::doPost(request,response) Pre: request.getParameter("email")!=null request.getParameter("password")!=null request.getParameter("nome")!=null request.getParameter("cognome")!=null request.getParameter("data")!=null
Post-condizioni	Context SigninServlet::doPost(request,response) Post: request.getSession().getAttribute("auth")!=null

Nome Classe	LogoutServlet
Descrizione	Questa classe è un control che si occupa di effettua il logout dell'utente dal sito.
Firma dei Metodi	# doPost(request: HttpServletRequest, response: HttpServletResponse):void # doGet(request: HttpServletRequest, response: HttpServletResponse):void
Pre-condizioni	Context LogoutServlet::doGet(request,response) Pre: request.getSession().getAttribute("auth")!=null
Post-condizioni	Context LogoutServlet::doGet(request,response) Post: request.getSession().getAttribute("auth")==null Response.sendRedirect("login.jsp")

Nome Classe	CategoriaServlet
Descrizione	Questa classe è un control che si occupa della visualizzazione degli occhiali in base alla scelta dell'utente.
Firma dei Metodi	# doPost(request: HttpServletRequest, response: HttpServletResponse):void # doGet(request: HttpServletRequest, response: HttpServletResponse):void
Pre-condizioni	Context CategoriaServlet::doGet(request,response) Pre: request.getParameter("tipo")!=null OR request.getParameter("sex")!=null
Post-condizioni	Context CategoriaServlet::doGet(request,response) Post: request.getAttribute("occhiali")!=null AND dispatcher!=null

Nome Classe	Checkout
Descrizione	Questa classe è un control che si occupa di passare le informazioni di un acquisto a OrdineDao, OcchialeOrdineDao e OcchialeDao.
Firma dei Metodi	# doPost(request: HttpServletRequest, response: HttpServletResponse):void # doGet(request: HttpServletRequest, response: HttpServletResponse):void
Pre-condizioni	Context Checkout::doGet(request,response) Pre: request.getParameter("sameadr")!=null request.getParameter("cardnumber")!=null carrello=request.getSession().getAttribute("carrello") !=null request.getSession().getAttribute("auth")!=null

Post-condizioni	Context Checkout::doGet(request,response) Post: carrello.dimension=0 AND dispatcher!=null AND request.getAttribute("address")!=null
-----------------	--

Nome Classe	EsisteEmail
Descrizione	Questa classe è un control che si occupa di passare la mail ad UtenteDao per vedere se esiste già in fase di registrazione.
Firma dei Metodi	# doPost(request: HttpServletRequest, response: HttpServletResponse):void # doGet(request: HttpServletRequest, response: HttpServletResponse):void
Pre-condizioni	Context EsisteEmail::doPost(request,response) Pre: request.getParameter("email")!=null
Post-condizioni	Context EsisteEmail::doPost(request,response) Post: request.getSession().getAttribute("email")!=null

Nome Classe	ForgetPassword
Descrizione	Questa classe è un control che si occupa di inserire una nuova password.
Firma dei Metodi	# doPost(request: HttpServletRequest, response: HttpServletResponse):void # doGet(request: HttpServletRequest, response: HttpServletResponse):void
Pre-condizioni	Context ForgetPassword::doPost(request,response) Post: request.getParameter("password")!=null AND request.getSession().getAttribute("email")!=null
Post-condizioni	Context ForgetPassword::doPost(request,response) Post: update password nel DB

Nome Classe	IndirizzoServlet
Descrizione	Questa classe è un control che si occupa di recuperare gli indirizzi usando IndirizziDao.
Firma dei Metodi	# doPost(request: HttpServletRequest, response: HttpServletResponse):void + doGet(request: HttpServletRequest, response: HttpServletResponse):void
Pre-condizioni	Context IndirizzoServlet::doGet(request,response) Pre: request.getSession().getAttribute("auth")!=null

Post-condizioni	Context IndirizzoServlet::doGet(request,response) Post: request.getAttribute("attivo")!=null AND request.getAttribute("indirizzi")!=null AND dispatcher!=null
Invariante	

Nome Classe	OcchialeControl
Descrizione	Questa classe è un control che si occupa di occupare di prelevare l'intero catalogo di Occhiali per visualizzarli nell'homepage.
Firma dei Metodi	# doPost(request: HttpServletRequest, response: HttpServletResponse):void # doGet(request: HttpServletRequest, response: HttpServletResponse):void
Pre-condizioni	Context OcchialeControl::doGet(request,response) Pre: request.getParameter("sort")!=null
Post-condizioni	Context OcchialeControl::doGet(request,response) Post: request.getAttribute("occhiali")!=null AND dispatcher!=null

Nome Classe	Search
Descrizione	Questa classe è un control che permette di ottenere determinati occhiali passando a OcchialiDao il brand scelto dall'utente.
Firma dei Metodi	# doPost(request: HttpServletRequest, response: HttpServletResponse):void # doGet(request: HttpServletRequest, response: HttpServletResponse):void
Pre-condizioni	Context Search::doGet (request,response) Pre: request.getParameter("brand")!=null
Post-condizioni	Context Search::doGet(request,response) Post: request.getAttribute("occhiali")!=null AND dispatcher!=null

Nome Classe	ServletAddAddress
Descrizione	Questa classe è un control che si occupa di passare un nuovo indirizzo a IndirizzoDao.
Firma dei Metodi	# doPost(request: HttpServletRequest, response: HttpServletResponse):void # doGet(request: HttpServletRequest, response: HttpServletResponse):void

Pre-condizioni	Context ServletAddAddress::doGet(request,response) Pre: request.getSession().getAttribute("auth")!=null request.getParameter("user_address")!=null request.getParameter("city")!=null request.getParameter("user_country")!=null request.getParameter("zip_code")!=null request.getParameter("user_phone")!=null
Post-condizioni	Context ServletAddAddress::doGet(request,response) Post: IndirizzoDao.doSave(idIndirizzo) eseguito and dispatcher!=null

Nome Classe	SearchCliente
Descrizione	Questa classe è un control che si occupa di passare i dati di un utente da cercare a UtenteDao e ottenere i suoi ordini, interfacciandosi con OrdineDao.
Firma dei Metodi	# doPost(request: HttpServletRequest, response: HttpServletResponse):void # doGet(request: HttpServletRequest, response: HttpServletResponse):void
Pre-condizioni	Context SearchCliente::doPost(request,response) Pre: request.getSession().getAttribute("auth")!=null request.getParameter("email")!=null
Post-condizioni	Context SearchCliente::doPost(request,response) Post: request.getAttribute("dati")!=null AND request.getParameter("ordini")!=null AND dispatcher!=null

Nome Classe	ListaOrdiniDataAdmin
Descrizione	Questa classe è un control che si occupa di passare i dati inseriti dall'admin ad OrdineDao per visualizzare gli ordini di una certa data.
Firma dei Metodi	# doPost(request: HttpServletRequest, response: HttpServletResponse):void # doGet(request: HttpServletRequest, response: HttpServletResponse):void
Pre-condizioni	Context ListaOrdiniDataAdmin::doGet(request,response) Pre: request.getSession().getAttribute("auth")!=null request.getParameter("skip")!=null request.getParameter("limit")!=null
Post-condizioni	Context ListaOrdiniDataAdmin::doGet(request,response)

	Post: request.getAttribute("ordini")!=null AND request.getAttribute("skip")!=null AND request.getAttribute("limit")!=null AND dispatcher!=null
--	--

Nome Classe	ServletAggiungiProdAdmin
Descrizione	Questa classe è un control che si occupa di passare i dati di un nuovo occhiale a OcchialeDao.
Firma dei Metodi	# doPost(request: HttpServletRequest, response: HttpServletResponse):void # doGet(request: HttpServletRequest, response: HttpServletResponse):void
Pre-condizioni	Context ServletAggiungiproAdmin::doPost(request,response) Pre: request.getSession().getAttribute("auth")!=null uploadPath!=null request. getPart("img1")!=null request. getPart("img2")!=null request. getParameter("id")!=null request. getParameter("nome")!=null request. getParameter("brand")!=null request. getParameter("prezzo")!=null request. getParameter("disp")!=null request. getParameter("colore")!=null request. getParameter("sesso")!=null request. getParameter("desc")!=null request. getParameter("categoria")!=null
Post-condizioni	Context ServletAggiungiproAdmin::doPost(request,response) Post: OcchialeDao.doSave(prodotto) eseguito AND dispatcher!=null

Nome Classe	ServletAmministratore
Descrizione	Questa classe è un control che si occupa di restituire un catalogo di occhiali.
Firma dei Metodi	# doPost(request: HttpServletRequest, response: HttpServletResponse):void # doGet(request: HttpServletRequest, response: HttpServletResponse):void
Pre-condizioni	Context ServletAmministratore::doGet(request,response) Pre: request.getSession().getAttribute("auth")!=null request. getParameter("id")!=null
Post-condizioni	Context ServletAmministratore::doGet(request,response) Post: request.getAttribute("admin")!=null request.getAttribute("dettagli")!=null OR

	request.getAttribute("modifica")!=null dispatcher!=null
--	--

Nome Classe	ServletCarrello
Descrizione	Questa classe è un control che si occupa di passare la quantità scelta dall'utente al Carrello.
Firma dei Metodi	# doPost(request: HttpServletRequest, response: HttpServletResponse):void # doGet(request: HttpServletRequest, response: HttpServletResponse):void
Pre-condizioni	Context ServletCarrello::doGet(request,response) Pre: request.getSession().getAttribute("carrello")!=null request. getParameter("id")!=null request. getParameter("scelta")!=null
Post-condizioni	Context ServletCarrello::doGet(request,response) Post: Carrello.insertQuantità(id,scelta) eseguito AND dispatcher!=null

Nome Classe	ServletControlloAdmin
Descrizione	Questa classe è un control che si occupa di gestire l'apertura delle 3 pagine azione dell'Admin.
Firma dei Metodi	# doPost(request: HttpServletRequest, response: HttpServletResponse):void # doGet(request: HttpServletRequest, response: HttpServletResponse):void
Pre-condizioni	Context ServletControlloAdmin::doGet(request,response) Pre: request.getSession().getAttribute("auth")!=null request. getParameter("id")!=null
Post-condizioni	Context ServletControlloAdmin::doGet(request,response) Post: request.getAttribute("admin")!=null AND dispatcher!=null
Invariante	

Nome Classe	ServletDettagliOrdine
Descrizione	Questa classe è un control che si occupa di recuperare i dettagli da OrdineDao tramite OcchialeOrdineDao.
Firma dei Metodi	# doPost(request: HttpServletRequest, response: HttpServletResponse):void # doGet(request: HttpServletRequest, response: HttpServletResponse):void
Pre-condizioni	Context ServletDettagliOrdine::doGet(request,response)

	Pre: request.getSession().getAttribute("auth")!=null request. getParameter("ordineId")!=null
Post-condizioni	Context ServletDettagliOrdine::doGet(request,response) Post: request.getAttribute("ordine")!=null request. getParameter("prodotti")!=null dispatcher!=null

Nome Classe	ServletEliminaProdAdmin
Descrizione	Questa classe è un control che si occupa di rimuovere passare l'id di un Occhiale a OcchialeDao, che eliminerà il prodotto dal DB.
Firma dei Metodi	# doPost(request: HttpServletRequest, response: HttpServletResponse):void # doGet(request: HttpServletRequest, response: HttpServletResponse):void
Pre-condizioni	Context ServletEliminaProdAdmin::doGet(request,response) Pre: request.getSession().getAttribute("auth")!=null request. getParameter("id")!=null
Post-condizioni	Context ServletEliminaProdAdmin::doGet(request,response) Post: Occhiale.doDelete(id) effettuato AND dispatcher!=null

Nome Classe	ServletEliminaProdotto
Descrizione	Questa classe è un control che si occupa di eliminare un occhiale dal Carrello, scelto dall'utente.
Firma dei Metodi	# doPost(request: HttpServletRequest, response: HttpServletResponse):void # doGet(request: HttpServletRequest, response: HttpServletResponse):void
Pre-condizioni	Context ServletEliminaProdotto::doGet(request,response) Pre: request.getSession().getAttribute("carrello")!=null request. getParameter("id")!=null
Post-condizioni	Context ServletEliminaProdotto::doGet(request,response) Post: request.getSession().getAttribute("carrello")!=null dispatcher!=null

Nome Classe	ServletModificaAmministratore
Descrizione	Questa classe è un control che si occupa di passare i dati nuovi di un Occhiale esistente a OcchialeDao che farà l'update dell'occhiale in questione.

Firma dei Metodi	# doPost(request: HttpServletRequest, response: HttpServletResponse):void # doGet(request: HttpServletRequest, response: HttpServletResponse):void
Pre-condizioni	Context ServletModificaAmministratore::doGet(request,response) Pre: request.getSession().getAttribute("auth")!=null request. getParameter("nome")!=null request. getParameter("descrizione")!=null
Post-condizioni	Context ServletModificaAmministratore::doGet(request,response) Post: OcchialeDao.doUpdate(occhiale) eseguito AND dispatcher!=null

Nome Classe	ServletOrdine
Descrizione	Questa classe è un control che si occupa di restituire e visualizzare la lista di ordini tramite OrdineDao,
Firma dei Metodi	# doPost(request: HttpServletRequest, response: HttpServletResponse):void # doGet(request: HttpServletRequest, response: HttpServletResponse):void
Pre-condizioni	Context ServletOrdine::doGet(request,response) Pre: request.getSession().getAttribute("auth")!=null
Post-condizioni	Context ServletOrdine::doGet(request,response) Post: request.getAttribute("ordini")!=null AND dispatcher!=null
Invariante	

Nome Classe	ServletProdotti
Descrizione	Questa classe è un control che si occupa di ottenere i dettagli di un occhiale passando l'id a OcchialeDao.
Firma dei Metodi	# doPost(request: HttpServletRequest, response: HttpServletResponse):void # doGet(request: HttpServletRequest, response: HttpServletResponse):void
Pre-condizioni	Context ServletProdotti::doGet(request,response) Pre: request.getParameter("id")!=null request.getParameter("azione")!=null
Post-condizioni	Context ServletProdotti::doGet(request,response) Post: request. getAttribute("carrello")!=null request. getAttribute("descrizione")!=null request. getAttribute("id")!=null dispatcher!=null

Nome Classe	ServletProfile
Descrizione	Questa classe è un control che si occupa di ottenere le informazioni dell'utente loggato, usando UtenteDao.
Firma dei Metodi	# doPost(request: HttpServletRequest, response: HttpServletResponse):void # doGet(request: HttpServletRequest, response: HttpServletResponse):void
Pre-condizioni	Context ServletProfile:doGet(request,response) Pre: request.getSession().getAttribute("auth")!=null
Post-condizioni	Context ServletProfile:doGet(request,response) Post: dispatcher!=null
Invariante	

Nome Classe	ServletCercaCliente
Descrizione	Questa classe è un control che si occupa di passare l'email a OrdineDao per ottenere la lista dei suoi ordini.
Firma dei Metodi	# doPost(request: HttpServletRequest, response: HttpServletResponse):void # doGet(request: HttpServletRequest, response: HttpServletResponse):void
Pre-condizioni	Context ServletCercaCliente:doGet(request,response) Pre: request.getSession().getAttribute("admin")!=null request.getParameter("email")!=null
Post-condizioni	Context ServletCercaCliente:doGet(request,response) Post: request.getAttribute("dati")!=null request.getParameter("ordini")!=null dispatcher!=null

Nome Classe	ShopControl
Descrizione	Questa classe è un control che si occupa della visualizzazione dell'intero catalogo.
Firma dei Metodi	# doPost(request: HttpServletRequest, response: HttpServletResponse):void # doGet(request: HttpServletRequest, response: HttpServletResponse):void
Pre-condizioni	Context ShopControl:doGet(request,response) Pre: request.getAttribute("sort")!=null
Post-condizioni	Context ShopControl:doGet(request,response) Post: request.getAttribute("occhiali")!=null AND dispatcher!=null

--	--

3.4 Extra

Nome Classe	CryptoPw
Descrizione	Questa classe è una classe Java che si occupa di crittografare le password.
Firma dei Metodi	+ digest(input: byte[]): byte[] + bytesToHex(input: byte[]): byte[] + isEqual(pw: String, pwStored:String): boolean + crypt(str: String): String
Pre-condizioni	Context CryptoPw::digest(input) Pre: input!=null Context CryptoPw:: bytesToHex (input) Pre: input!=null Context CryptoPw::isEqual (pw,pwStored) Pre: pw!=null and pwStored!=null Context CryptoPw:: cryptPw(str) Pre: str!=null
Post-condizioni	Context CryptoPw::digest(input) Post: result!=null Context CryptoPw::bytesToHex(input) Post: sb.toString()!=null (sb è uno StringBuilder) Context CryptoPw::isEqual(pw, pwStored) Post: true if CryptoPw.crypt(pw)== pwStored, false altrimenti Context CryptoPw::cryptPw(str) Post: bytesToHex(CryptoPw.digest(str.getBytes(<i>UTF_8</i>)))!=null
Invariante	

Glossario

Nome	Definizione
Occhiale	Un occhiale memorizzato nel Sistema I cui dati possono essere visualizzati dal sito
Ospite	Un utilizzatore del sito non loggato, può visualizzare l'homepage e il catalogo completo,

	ed aggiungere al carrello I prodotti
Utente	Un utilizzatore del sito loggato, può acquistare i prodotti, visualizzare i suoi ordini
Admin	Utente speciale del sito loggato, può rimuovere e modificare prodotti del catalogo ed aggiungerne di nuovi, visualizzare ordini degli altri utenti,