Object Design Document

GreenBottle

Riferimento	2025_C09_ODD
Versione	1.0
Data	11/12/2024
Destinatario	Top management
Presentato da	G. Ruocco, P. D'Antuono, S. Cozzolino, F. C. Ponticelli, G. Pastena, P. Muraca, M. Marino
Approvato da	F. M. Puca, L. Sorrentino

Team Members

Nome	Ruolo	Acronimo	Informazioni di contatto
Francesco Maria Puca	Project Manager	FMP	f.puca3@studenti.unisa.it
Lorenzo Sorrentino	Project Manager	LS	l.sorrentino66@studenti.unisa.it
Giovanni Ruocco	Team Member	GR	g.ruocco45@studenti.unisa.it
Pietro D'Antuono	Team Member	PD	p.dantuono2@studenti.unisa.it
Stefano Cozzolino	Team Member	STC	s.cozzolino15@studenti.unisa.it
Fabio Catello Ponticelli	Team Member	FCP	f.ponticelli2@studenti.unisa.it
Giuseppe Pastena	Team Member	GP	g.pastena1@studenti.unisa.it
Pasquale Muraca	Team Member	PM	p.muraca@studenti.unisa.it
Michael Marino	Team Member	MM	m.marino107@studenti.unisa.it
Salvatore Conte	Team Member	SAC	s.conte19@studenti.unisa.it

Revision History

Data	Versione	Descrizione	Autori
27/11/2024	0.1	Introduzione ODD	FCP, MM
27/11/2024	0.1.1	Aggiunta la sezione "Object Design Trade-offs"	FCP, MM
27/11/2024	0.1.2	Aggiunta la sezione "Componenti off the Shelf"	FCP, MM
27/11/2024	0.1.3	Aggiunta la sezione "Linee guida"	FCP, MM
27/11/2024	0.1.4	Aggiunta la sezione "Definizioni, acronimi e abbreviazione"	FCP, MM
27/11/2024	0.1.5	Aggiunta la sezione "Riferimenti"	FCP, MM
27/11/2024	0.2	Aggiunta la sezione "Class Diagram"	STC, GP, PM
29/11/2024	0.3	Aggiunta la sezione "Packages"	TUTTI
06/12/2024	0.4	Aggiunta la sezione "Design Patterns"	TUTTI
06/12/2024	0.5	Aggiunta la sezione "Class Interfaces"	TUTTI
11/12/2024	0.6	Aggiunta la sezione "Glossario"	GR, FCP, MM
11/12/2024	1.0	Revisione	GR, FCP, MM

Summary

1 Introduzione 5
1.1 Object Design Trade-offs 5
1.1.1 Componenti off the Shelf 5
1.2 Linee guida per la documentazione dell'interfaccia 7
1.3 Definizione, acronimi e abbreviazioni 7
1.4 Riferimenti 7
2 Packages 8
2.1 Package GreenBottle 8
2.1.1 Package Totale 9
2.1.2 Package Application Logic 10
2.1.3 Package Storage 13
3 Class Interfaces 16
3.1 Package Accesso Control 16
3.2 Package Catalogo Control 19
3.3 Package Abbonamento Control 21
3.4 Package Area Personale Control 23
3.5 Package Ordine Control 25
3.6 Package Ottimizzazione Consegne Control 29
4 Class Diagram 31
5 Design Patterns 32
5.1 Adapter 32
5.2 Builder Pattern 33
6 Glossario 34

1 Introduzione

GreenBottle si occupa della distribuzione di bevande, con un forte impegno verso la sostenibilità ambientale, facilitando un ciclo di riuso sostenibile attraverso i suoi servizi di consegna e ritiro.

A tale scopo, per incrementare il bacino di utenza di queste piccole realtà, si propone di creare uno strumento che migliori l'esperienza, semplificando i processi di consegna e ritiro di bottiglie di vetro permettendo il tracciamento e la programmazione di quest'ultime, offrendo supporto speciale per persone con difficoltà motorie.

In questa prima sezione del documento verranno descritti i trade-offs individuati e componenti off-the-shelf usati.

1.1 Object Design Trade-offs

Durante la fase di analisi e di progettazione sono stati individuati diversi compromessi per lo sviluppo del sistema. Inoltre, anche nella fase dell'Object Design sorgono diversi trade-offs di progettazione analizzati nel corso di questa sezione del documento:

Readability vs. Release Time

Il tempo di rilascio della piattaforma rispetto a una miglior leggibilità del codice è uno dei primi obiettivi prioritari individuati. Si è deciso di prioritizzare il tempo di rilascio in quanto commentare ogni singola riga di codice porterebbe un enorme consumo di tempo rallentando così il rilascio della piattaforma. Nonostante ciò, si propone di rendere il codice il più leggibile e mantenibile possibile.

Buy vs. Build

Abbiamo pensato di riutilizzare il più possibile delle soluzioni off-the-shelf in quanto riscrivere del codice da zero porterebbe, oltre ad un consumo di tempo e risorse, anche a potenziali errori che potrebbero impiegare ulteriore tempo. Quindi, si è deciso di utilizzare delle soluzioni precostruite per lo sviluppo di applicazioni web, scegliendoli in maniera avveduta al fine di trarne maggior vantaggio possibile da essi.

1.1.1 Componenti off the Shelf

Per facilitare e velocizzare lo sviluppo della piattaforma GreenBottle, saranno utilizzate diverse componenti off-the-shelf di seguito riportate:

jQuery e AJAX

Per permettere all'interfaccia grafica di rispondere alle azioni dell'utente in modo rapido ed efficiente e per migliorare la user experience durante l'interazione dell'utente con la piattaforma verranno utilizzati jQuery e AJAX.

MySql

MySQL Connector è il driver che permette l'interoperabilità fra Java e MySQL. Viene impiegato per permettere alla WebApp di interfacciarsi col Database, e dunque compiere operazioni CRUD.

Bootstrap

Per permettere lo sviluppo di un'interfaccia grafica che sia responsive ed esteticamente piacevole, per la maggior parte degli utenti, sarà utilizzato il framework Bootstrap.

Spring Data JPA

Basato su tecnologia ORM (Object-Relational Mapping), ci permette di interrogare e gestire le tabelle di un database senza dover necessariamente scrivere query, ma utilizzando classi, oggetti e metodi. Inoltre, in caso di cambiamento di dbms, non avendo scritto query, non bisogna preoccuparsi di adattarle al nuovo DBMS, farà tutto il framework.

String Data JPA ci servirà per:

- Generare in automatico il codice per la creazione delle tabelle
- Implementare le operazioni crud
- implementare query complesse
- Gestire le relazioni tra tabelle

Spring Web MVC

Framework web di Spring basato su Servlet-API di Java. Nonostante il nome possa essere fuorviante, non sarà utilizzata l'architettura MVC.

Spring Boot

Spring Boot è una soluzione "convention over configuration" per il framework Spring di Java, che riduce la complessità di configurazione di nuovi progetti Spring.

Spring Boot Thymeleaf

Spring Boot Thymeleaf è una soluzione "convention over configuration" per lo sviluppo web con Spring, semplificando la gestione delle viste HTML dinamiche.

1.2 Linee guida per la documentazione dell'interfaccia

Le linee guida includono una lista di regole che gli sviluppatori dovrebbero rispettare durante la progettazione delle interfacce.

Di seguito una lista di link alle convenzioni usate per definire le linee guida:

- Java: Link alla convenzione
- HTML/CSS: Link alla convenzione
- JS: Link alla convenzione

1.3 Definizione, acronimi e abbreviazioni

- **DP**: Design Pattern
- **RAD:** Requirements Analysis Document
- **SDD:** System Design Document
- **ODD:** Object Design Document
- **SOW:** Statement Of Work

1.4 Riferimenti

Di seguito una lista di riferimenti ad altri documenti utili durante la lettura:

- Documento 2025_C09_RAD
- Documento 2025_C09_SDD
- Documento 2025_C09_ODD

2 Packages

La struttura del progetto ricalca la struttura di directory standard definita da Maven e Spring.

- **src,** contiene tutti i file sorgente
  - **main**
    - **java,** contiene le classi Java relative alle componenti Application Logic e Storage
    - **resources,** contiene i file relativi alla componente Interface
      - **static,** contiene i fogli di stile CSS e gli script JS
      - **templates,** contiene i file HTML reindirizzati da Thymeleaf
  - **test,** contiene il codice necessario alla fase di testing
    - **java,** contiene le classi Java per l'implementazione del testing
- **pom.xml,** contiene la configurazione del progetto Maven

2.1 Package GreenBottle

La divisione in packages viene eseguita osservando la suddivisione in sottosistemi effettuata in fase di System Design, cosa che implica:

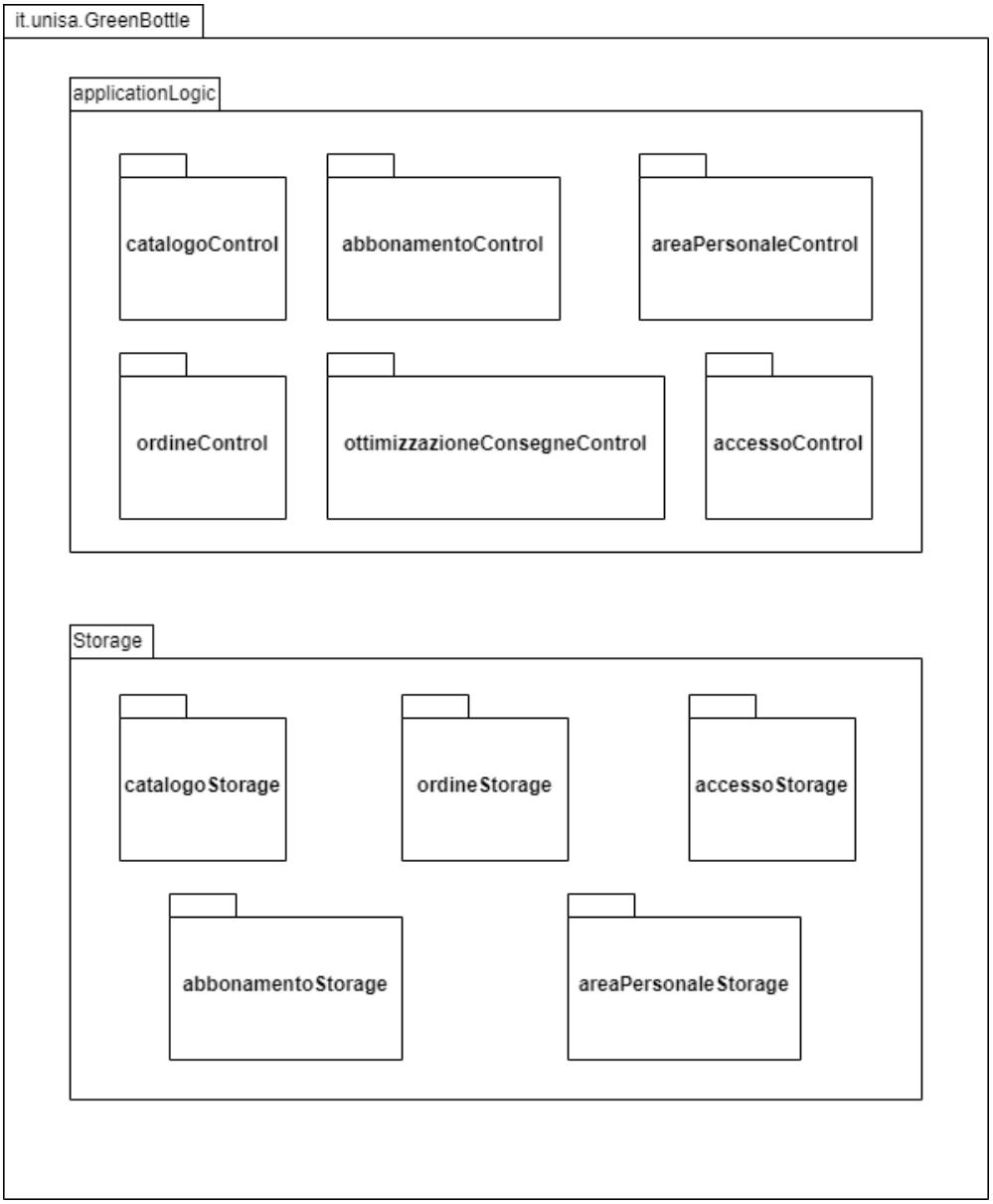
- Un uso evidente dell'architettura Three-Tier;
- Che le dipendenze tra i packages corrispondano a quelle indicate nel SDD.

La struttura dei package di GreenBottle è ottenuta a partire da due scelte principali:

1. Creare due package separati per i layer di Application Logic e Storage, che sono a loro volta divisi internamente in base ai singoli sottosistemi.
2. Suddividere ulteriormente i sotto-package di Storage in package Dao ed Entity per separare la logica di accesso al database dalle entità che mappano le rispettive tabelle.

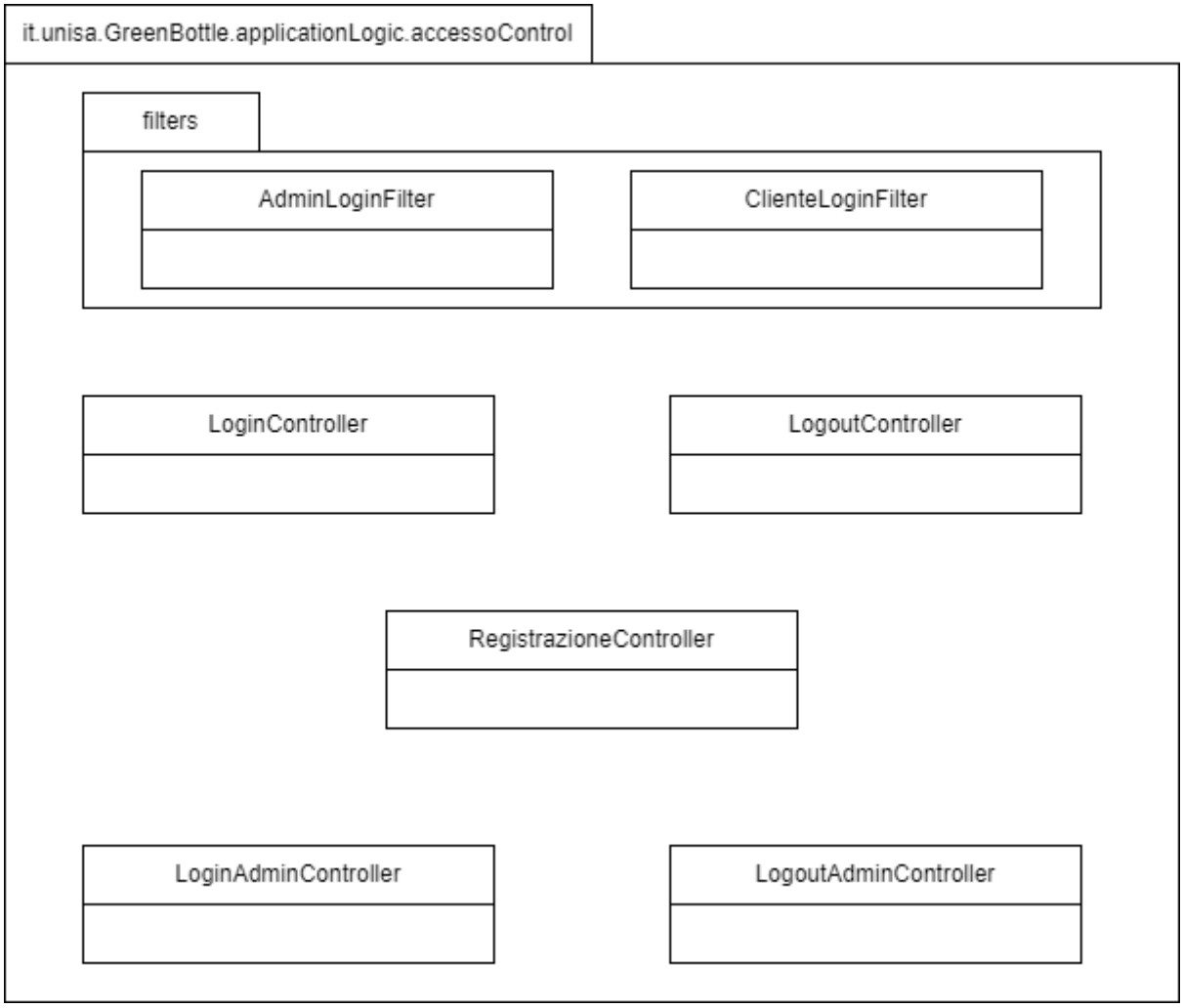
Alcune classi e packages di utility sono stati omessi nei seguenti schemi a scopo di leggibilità, perché non fondamentali per comprendere la struttura del progetto. Questi sono comunque visionabili nel JavaDoc.

2.1.1 Package Totale

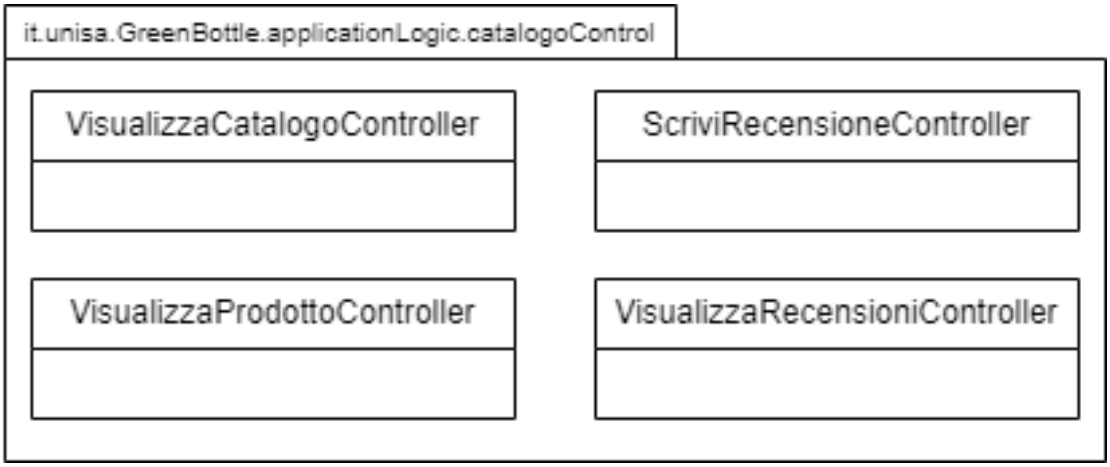


2.1.2 Package Application Logic

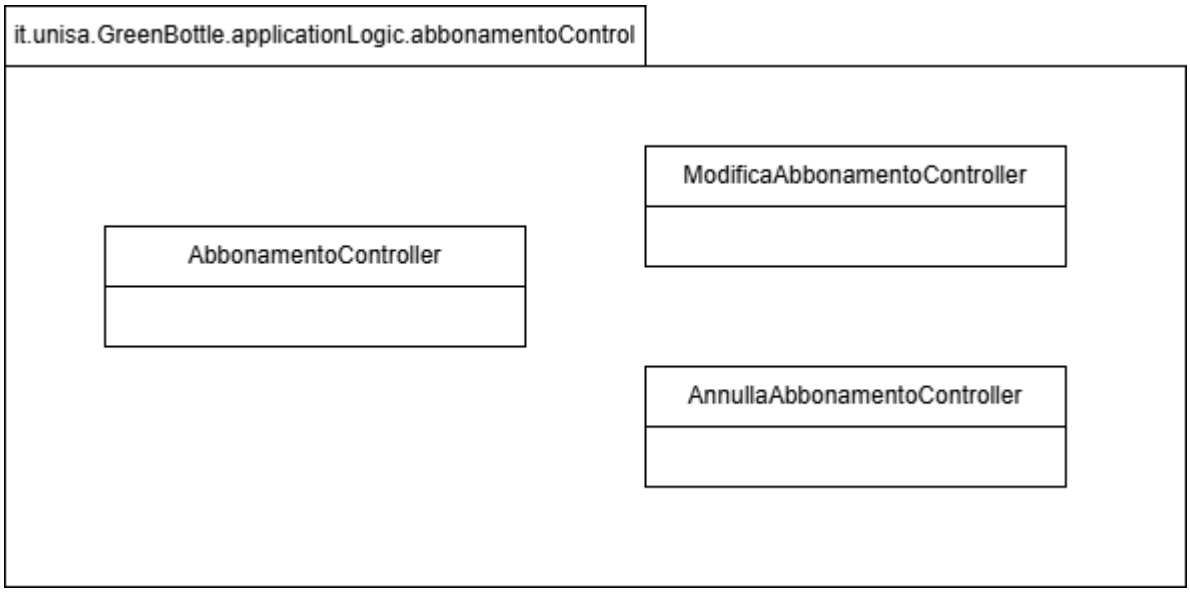
Package Accesso Control



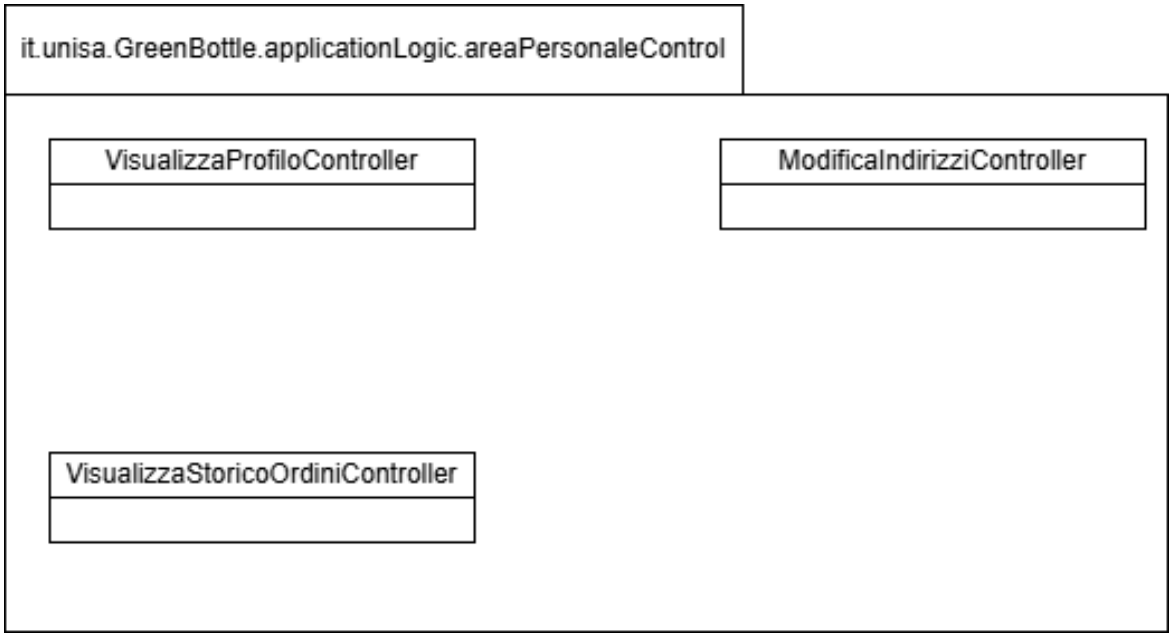
Package Catalogo Control



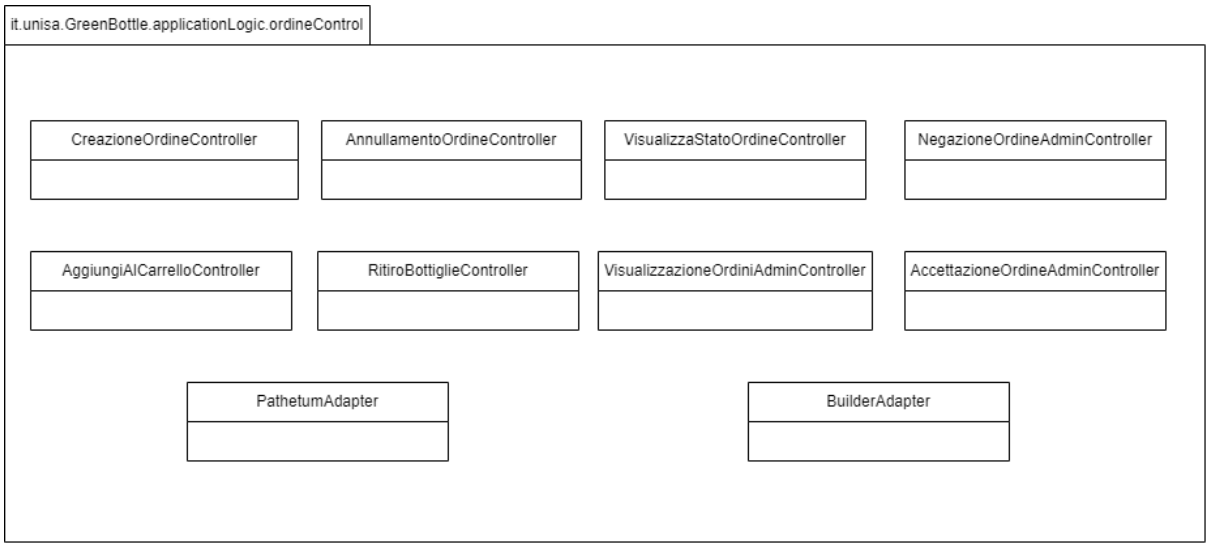
Package Abbonamento Control



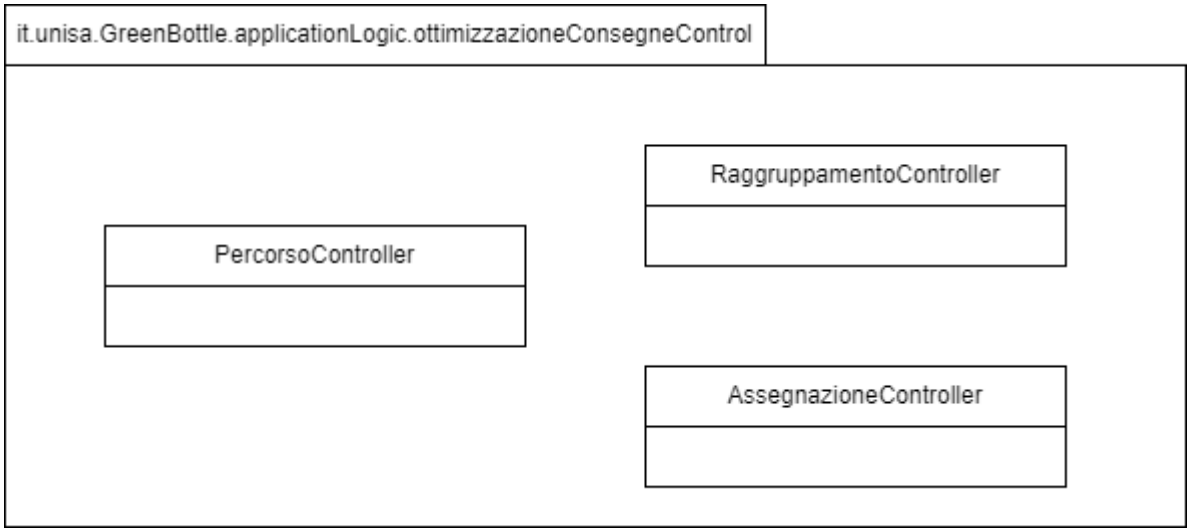
Package Area Personale Control



Package Ordine Control

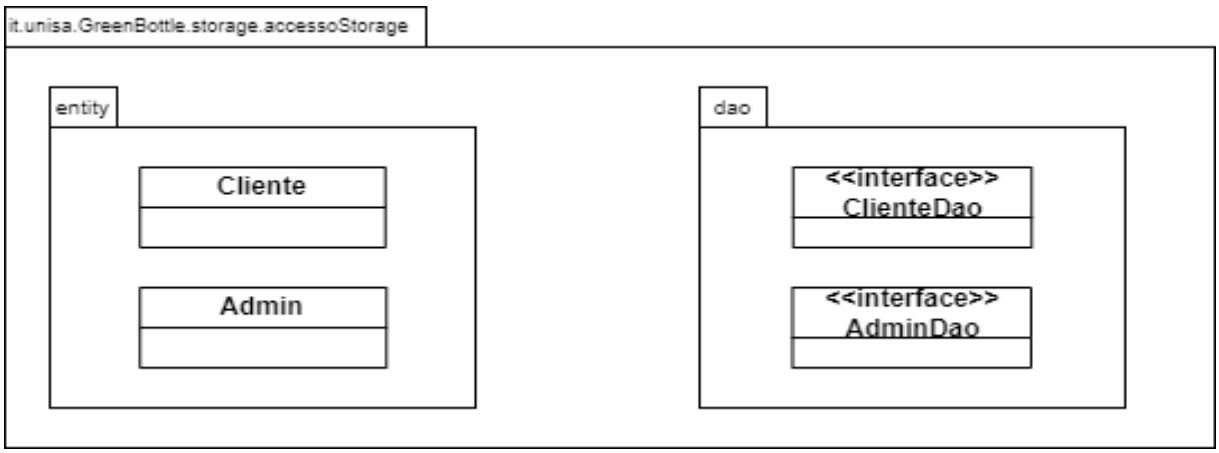


Package Ottimizzazione Consegne Control

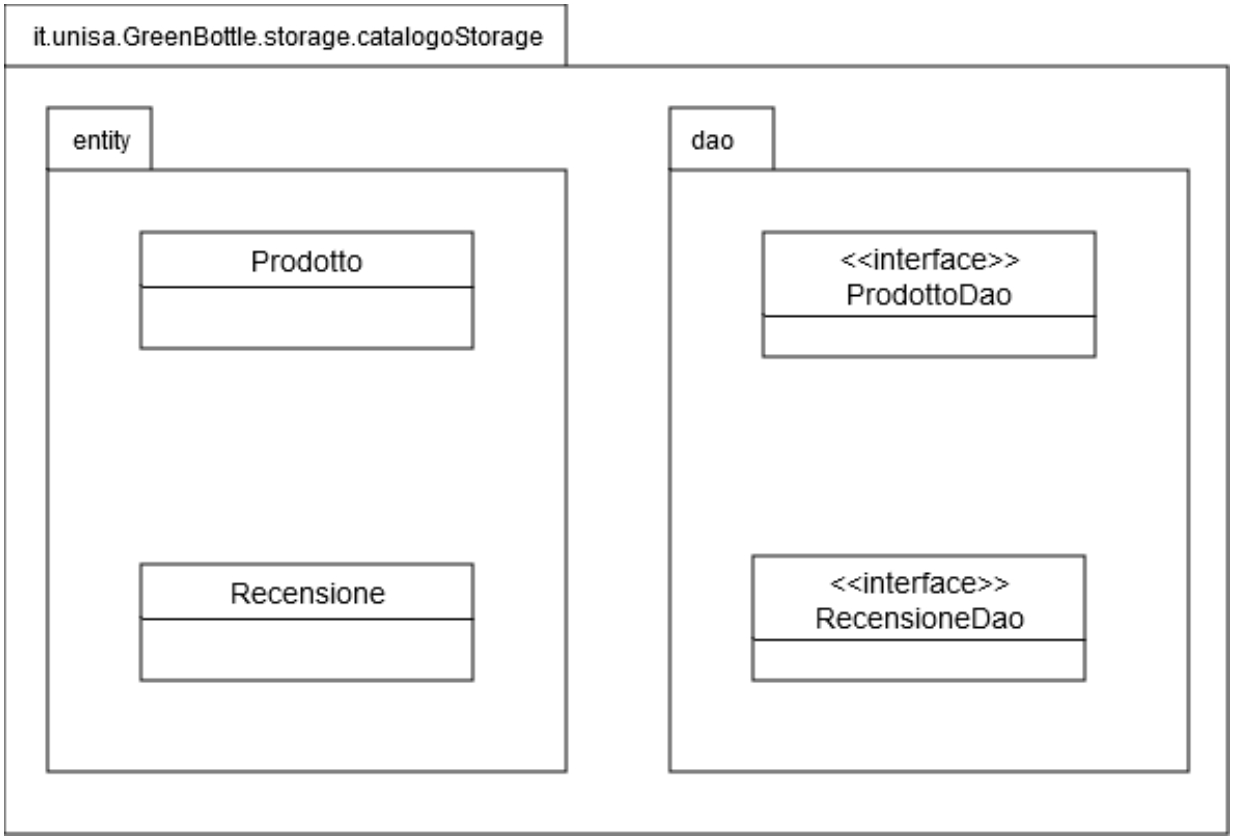


2.1.3 Package Storage

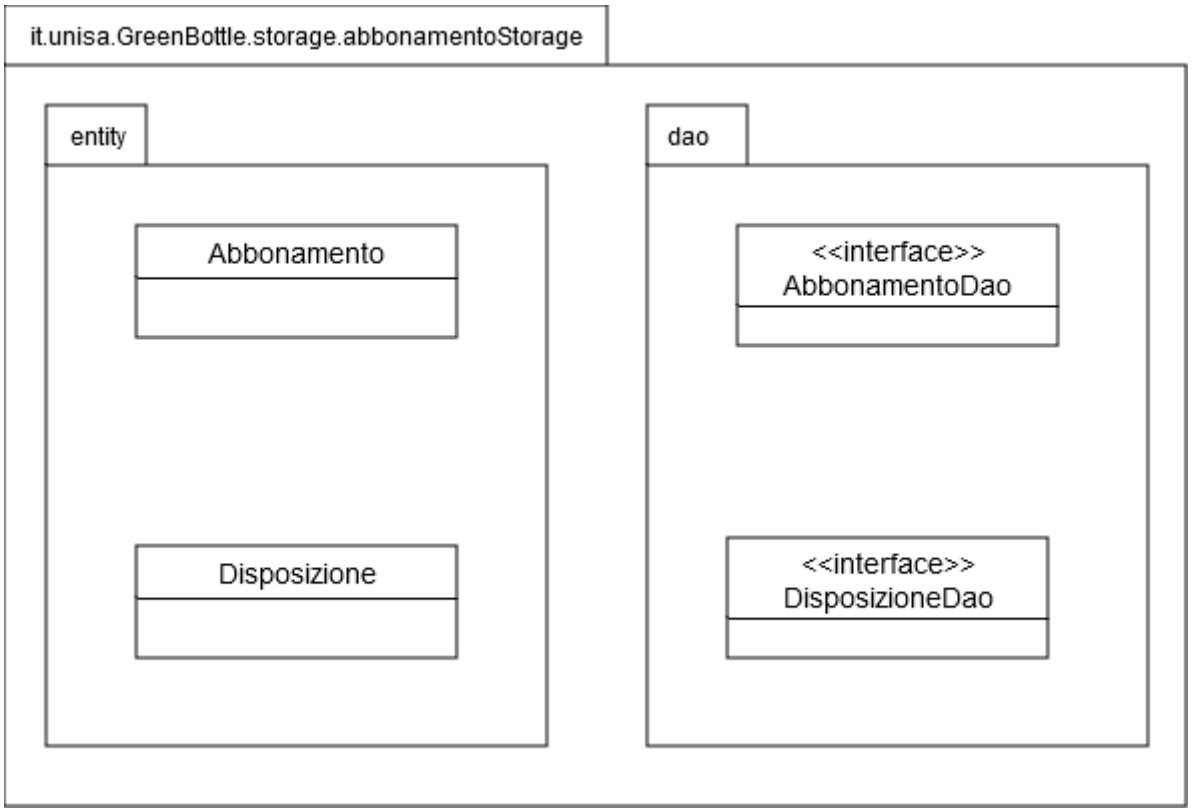
Package Accesso Storage



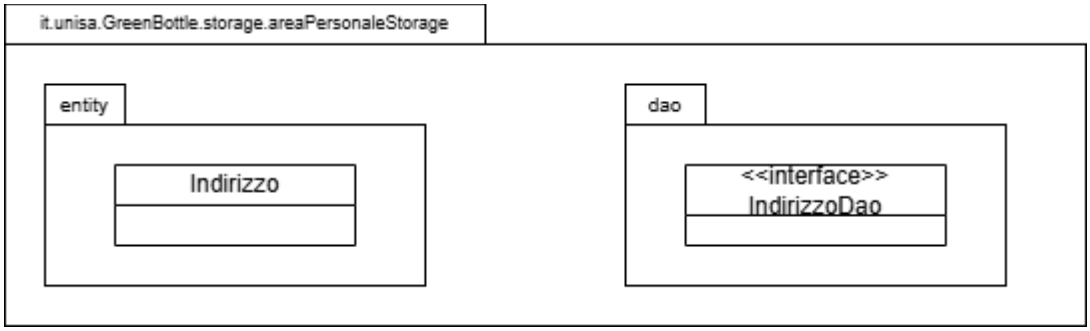
Package Catalogo Storage



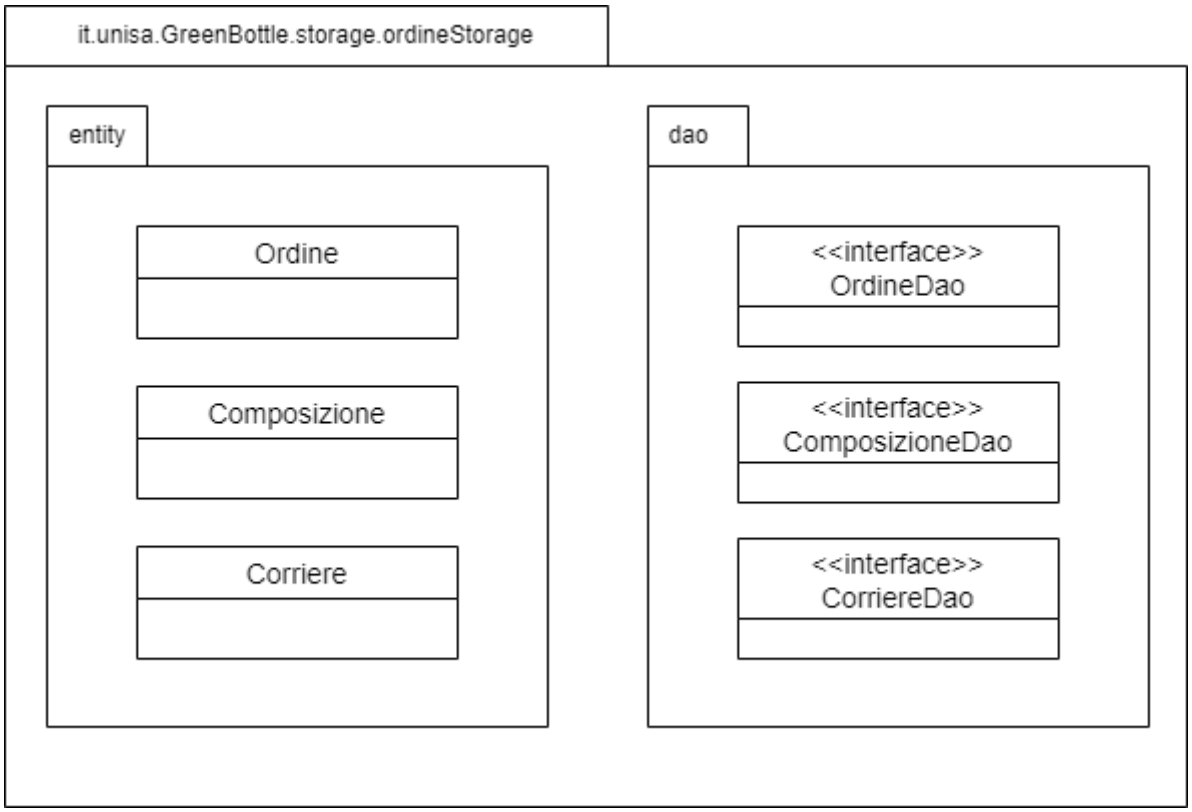
Package Abbonamento Storage



Package Area Personale Storage



Package Ordine Storage



3 Class Interfaces

3.1 Package Accesso Control

Nome classe	LoginController
Descrizione	Implementa il controller per il login del Cliente.
Metodi	+ get(LoginForm loginForm, Model model):String + post(LoginForm loginForm, BindingResult bindingResult, Model model, HttpSession session):String
Invariante della classe	/

Nome metodo	+ get(LoginForm loginForm, Model model):
Descrizione	Implementa la funzionalità di smistare il Cliente sulla view di autenticazione View/login.
Precondizione	
Postcondizione	
Invariante della classe	/
Nome metodo	+ post(LoginForm loginForm, BindingResult bindingResult, Model model, HttpSession session):String
Descrizione	Implementa la funzionalità di login di un Cliente
Precondizione	context: LoginController::post(loginForm, bindingResult, model, session) pre: not bindingResult.hasErrors()
Postcondizione	context: LoginController::post(loginForm, bindingResult, model, session) post: sessionCliente.setCliente(cliente)
Invariante della classe	/

Nome classe	LogoutController
Descrizione	Implementa il controller per il logout per l'Utente
Metodi	+ get(HttpSession session):String
Invariante della classe	/

Nome metodo	+ get(HttpSession session):
Descrizione	Implementa la funzionalità di logout di un Utente
Precondizione	
Postcondizione	context:LogoutController::get(HttpSession session) Post:sessionCliente.getCliente().isEmpty() && sessionAdmin.getAdmin().isEmpty()
Invariante della classe	/

Nome classe	RegistrazioneController
Descrizione	Implementa il controller per la registrazione del Cliente.
Metodi	+ get(RegistrazioneForm registrazioneForm):String + post(RegistrazioneForm registrazioneForm, BindingResult bindingResult, Model model, HttpServletResponse httpServletResponse): String
Invariante della classe	/

Nome metodo	+ get(RegistrazioneForm registrazioneForm):
Descrizione	Implementa la funzionalità di smistare sulla view di autenticazioneView/registrazione.
Precondizione	
Postcondizione	
Invariante della classe	/
Nome metodo	+ post(RegistrazioneForm registrazioneForm, BindingResult bindingResult, Model model, HttpServletResponse httpServletResponse):
Descrizione	Implementa la funzionalità di registrazione di un Cliente
Precondizione	context: RegistrazioneController::post(registrazioneForm, bindingResult) pre: not bindingResult.hasErrors()
Postcondizione	context: RegistrazioneController::post(registrazioneForm, bindingResult, HttpServletResponse httpServletResponse) post: ClienteDao.save(cliente) == true.
Invariante della classe	/

Nome classe	LoginAdminController
Descrizione	Implementa il controller per il login per gli Amministratori.
Metodi	+ get(LoginForm loginForm, Model model):String + post(LoginForm loginForm, BindingResult bindingResult, Model model, HttpSession HttpSession):String
Invariante della classe	/

Nome metodo	+ get(LoginForm loginForm, Model model):
Descrizione	Implementa la funzionalità di smistare l'Amministratore sulla view di autenticazioneView/login.
Precondizione	
Postcondizione	
Invariante della classe	/
Nome metodo	+ post(LoginForm loginForm, BindingResult bindingResult, Model model, HttpServletResponse httpServletResponse):
Descrizione	Implementa la funzionalità di login di un Amministratore
Precondizione	context: LoginAdminController::post(loginForm, bindingResult, model) pre: not bindingResult.hasErrors()
Postcondizione	context: LoginAdminController::post(loginForm, bindingResult, mode, sessionl) post: sessionAdmin.setAdmin(admin)
Invariante della classe	/

3.2 Package Catalogo Control

Nome classe	VisualizzaCatalogoController
Descrizione	Implementa il controller della visualizzazione del catalogo.
Metodi	+ get(FiltroForm filtroForm, BindingResult bindingResult, Model model, HttpServletResponse httpServletResponse): String
Invariante della classe	/

Nome metodo	+ get(FiltroForm filtroForm, BindingResult bindingResult, Model model, HttpServletResponse httpServletResponse): String
Descrizione	Implementa la funzionalità di recupero dell'elenco dei prodotti nel catalogo. Se viene fornito un filtro, applica il filtro per restituire solo i prodotti corrispondenti. Se il filtro non è fornito, restituisce l'intero catalogo.
Precondizione	
Postcondizione	Context: VisualizzaCatalogoController::get(model, filtro) Post: model.containsAttribute(prodotti) && model.getAttribute(prodotti).equals(filtro.isPresent() ? prodottiFiltrati : catalogoCompleto)
Invariante della classe	/

Nome classe	ScriviRecensioneController
Descrizione	Implementa il controller della scrittura di una recensione.
Metodi	+ post(Model model, Long idProdotto, Long idCliente): String
Invariante della classe	/

Nome metodo	+ post(Model model, Long idProdotto, Long idCliente): String
Descrizione	Implementa la funzionalità di scrittura di una recensione.
Precondizione	Context: ScriviRecensioneController::post(model, prodotto, cliente) Pre: RecensioneDao.existsByUserAndProduct(prodotto, cliente) == null Pre: sessionCliente.getCliente().isPresent()
Postcondizione	Context: VisualizzaCatalogoController::post(model) Post: RecensioneDao.save(recensione) == true Post: model.containsAttribute("recensione") && !recensione.isEmpty()
Invariante della classe	/

Nome classe	VisualizzaProdottoController
Descrizione	Implementa il controller della visualizzazione dettagliata di un prodotto.
Metodi	+ get(Long id): ModelAndView
Invariante della classe	/

Nome metodo	+ get(Long id): ModelAndView
Descrizione	Implementa la funzionalità di smistare l'utente tra catalogo e prodotto.
Precondizione	Context: VisualizzaProdottoController::get(id) Pre: prodottoDao.existsById(id)
Postcondizione	Context: VisualizzaProdottoController::get(id) Post: result.getObject(prodotto) != null
Invariante della classe	/

Nome classe	VisualizzaRecensioniController
Descrizione	Implementa il controller della visualizzazione delle proprie recensioni.
Metodi	+ get(Model model, Long idCliente): ModelAndView
Invariante della classe	/

Nome metodo	+ get(Model model, Long idCliente): ModelAndView
Descrizione	Implementa la funzionalità di visualizzazione dello storico delle recensioni di un Cliente.
Precondizione	Context: VisualizzaRecensioniController::get(model, id) Pre: sessionCliente.getCliente().isPresent() Pre: ClienteDao.existsById(id)
Postcondizione	Context: VisualizzaProdottoController::get(id) Post: model.containsAttribute(recensioni) Post: sessionCliente.getCliente().isPresent()
Invariante della classe	/

3.3 Package Abbonamento Control

Nome classe	AbbonamentoController
Descrizione	Implementa il controller per la visualizzazione dei dettagli degli abbonamenti e la sottoscrizione degli abbonamenti
Metodi	+ get(Model model, String tipo, Long id): ModelAndView + post(Model model, BindingResult binding Result, AbbonamentoForm abbonamentoForm, HttpServletRespone httpServletResponse): String
Invariante della classe	/

Nome metodo	+ get(Model model, String tipo, Long id): ModelAndView
Descrizione	Implementa la funzionalità di visualizzazione dei dettagli dell'abbonamento
Precondizione	Context: AbbonamentoController::get(model, tipo, id)
Postcondizione	Context: AbbonamentoController::get(model, tipo, id)
Invariante della classe	/

Nome metodo	+ post(Model model, BindingResult binding Result, AbbonamentoForm abbonamentoForm, HttpServletRespone httpServletResponse): String
Descrizione	Implementa la funzionalità di sottoscrizione di un abbonamento
Precondizione	Context: AbbonamentoController::post(model, bindingResult, abbonamentoForm, httpServletResponse) Pre: sessionCliente.getCliente().isPresent() Pre: ClienteDao.existsById(id) Pre: AbbonamentoDao.existsById(idAbbonamento)
Postcondizione	Context: AbbonamentoController::post(model, bindingResult, abbonamentoForm, httpServletResponse) Post: model.containsAttribute("abbonamento") Post: ClienteDao.saveAbbonamento(idAbbonamento)
Invariante della classe	/

Nome classe	ModificaAbbonamentoController
Descrizione	Implementa il controller per la modifica dell'abbonamento di un Cliente.
Metodi	+ post(Model model, Long idAbbonamento): String
Invariante della classe	/

Nome metodo	+ post(Model model, Long idAbbonamento): String
Descrizione	Implementa la funzionalità di modifica dell'abbonamento di un Cliente.
Precondizione	Context: VisualizzaAbbonamentoController::get(model, id) Pre: sessionCliente.getCliente().isPresent() Pre: AbbonamentoDao.existsById(idAbbonamento)
Postcondizione	Context: VisualizzaAbbonamentoController::get(model, id) Post: ClienteDao.updateAbbonamento(idAbbonamento) Post: model.containsAttribute(abbonamento)
Invariante della classe	/

Nome classe	AnnullaAbbonamentoController
Descrizione	Implementa il controller per annullare la sottoscrizione a un abbonamento.
Metodi	+ post(Model model, Long idAbbonamento): String
Invariante della classe	/

Nome metodo	+ post(Model model): String
Descrizione	Implementa la funzionalità di annullamento.
Precondizione	Context: VisualizzaAbbonamentoController::get(model, id) Pre: sessionCliente.getCliente().isPresent()
Postcondizione	Context: VisualizzaAbbonamentoController::get(model, id) Post: ClienteDao.deleteAbbonamento()
Invariante della classe	/

3.4 Package Area Personale Control

Nome classe	VisualizzaProfiloController
Descrizione	Implementa il controller per la visualizzazione dei dettagli del profilo Cliente.
Metodi	+ get(Model model, Long idCliente): ModelAndView
Invariante della classe	/

Nome metodo	+ get(Model model, Long idCliente): ModelAndView
Descrizione	Implementa la visualizzazione profilo.
Precondizione	Context: VisualizzaProfiloController::get(model, id) Pre: sessionCliente.getCliente().isPresent() Pre: ClienteDao.existsById(id)
Postcondizione	Context: VisualizzaProfiloController::get(model, id) Post: model.containsAttribute(cliente) Post: sessionCliente.getCliente().isPresent()
Invariante della classe	/

Nome classe	ModificaIndirizziController
Descrizione	Implementa il controller per la modifica o aggiunta di indirizzi per il Cliente.
Metodi	+ post(Model model, Long idIndirizzo): String + post(Model model, Long idIndirizzo, Long idCliente): String
Invariante della classe	/

Nome metodo	+ post(Model model, Long idIndirizzo): String
Descrizione	Implementa la funzionalità di modifica di un indirizzo di un Cliente.
Precondizione	Context: ModificaIndirizziController::get(model, id) Pre: sessionCliente.getCliente().isPresent() Pre: IndirizzoDao.existsById(idIndirizzo)
Postcondizione	Context: ModificaIndirizziController::get(model, id) Post: ClienteDao.updateIndirizzo(idIndirizzo, idIndirizzo2) Post: model.containsAttribute(indirizzo)
Invariante della classe	/

Nome metodo	+ post(Model model, Long idIndirizzo): String
Descrizione	Implementa la funzionalità di aggiunta di un nuovo indirizzo.
Precondizione	Context: ModificaIndirizziController::get(model, id) Pre: sessionCliente.getCliente().isPresent() Pre: IndirizzoDao.existsById(idIndirizzo)
Postcondizione	Context: ModificaIndirizziController::get(model, id) Post: ClienteDao.addIndirizzo(idIndirizzo) Post: model.containsAttribute(indirizzo)
Invariante della classe	/

Nome classe	VisualizzaStoricoOrdiniController
Descrizione	Implementa il controller per la visualizzazione dello storico degli Ordini del Cliente.
Metodi	+ get(DataForm dataForm, Model model): ModelAndView
Invariante della classe	/

Nome metodo	+ get(DataForm dataForm, Model model): ModelAndView
Descrizione	Implementa la funzionalità di visualizzazione dello storico degli ordini di un Cliente.
Precondizione	Context: VisualizzaStoricoOrdiniController::get(model, id) Pre: sessionCliente.getCliente().isPresent() Pre: ClienteDao.existsById(id)
Postcondizione	Context: VisualizzaStoricoOrdiniController::get(id) Post: model.containsAttribute(ordini) Post: sessionCliente.getCliente().isPresent()
Invariante della classe	/

3.5 Package Ordine Control

Nome classe	CreazioneOrdineController
Descrizione	Implementa la funzionalità di creazione dell'ordine di un Cliente.
Metodi	+ get(Model model, HttpServletResponse httpServletResponse) : String + post(OrdineForm ordineForm, BindingResult bindingResult, Model model, HttpServletResponse httpServletResponse) : String
Invariante della classe	/

Nome metodo	+ get(Model model, HttpServletResponse httpServletResponse) : String
Descrizione	Implementa la funzionalità di visualizzazione della pagina di Checkout per un ordine.
Precondizione	Context: CreazioneOrdineController::get(model, httpServletResponse) Pre: sessionCarrello.getRealCarrello().isPresent() Pre: sessionCliente.getCliente().isPresent()
Postcondizione	/
Invariante della classe	/

Nome metodo	+ post(OrdineForm ordineForm, BindingResult bindingResult, Model model, HttpServletResponse httpServletResponse) : String
Descrizione	Implementa la funzionalità di creazione di un nuovo ordine per il Cliente.
Precondizione	Context: CreazioneOrdineController::post(ordineForm, bindingResult, model, httpServletResponse) Pre: sessionCarrello.getRealCarrello().isPresent() Pre: sessionCliente.getCliente().isPresent() Pre: @Valid
Postcondizione	Context: CreazioneOrdineController::post(carta, indirizzoSpedizione, model) Post: OrdineDao.save(ordine) == true Post: model.containsAttribute(ordine)
Invariante della classe	/

Nome classe	AnnullamentoOrdineController
Descrizione	Implementa la funzionalità di annullamento dell'ordine di un Cliente.
Metodi	+ post(Long idOrdine, Model model): String
Invariante della classe	/

Nome metodo	+ post(Long idOrdine, Model model): String
Descrizione	Implementa la funzionalità di annullare un ordine esistente di un Cliente.
Precondizione	Context: AnnullamentoOrdineController::post(idOrdine, model) Pre: sessionCliente.getCliente().isPresent() Pre: OrdineDao.existsById(idOrdine)
Postcondizione	Context: AnnullamentoOrdineController::post(idOrdine, model) Post: OrdineDao.delete(idOrdine) == true Post: model.Ordine.existsByID(idOrdineAnnullato) == false
Invariante della classe	/

Nome classe	AggiungiAlCarrelloController
Descrizione	Implementa la funzionalità di aggiungere un prodotto al carrello di un Cliente.
Metodi	+ post(Long idProdotto, Integer quantita, Model model): String
Invariante della classe	/

Nome metodo	+ post(Long idProdotto, Integer quantita, Model model): String
Descrizione	Aggiunge un prodotto con una quantita specifica al carrello del Cliente.
Precondizione	Context: AggiungiAlCarrelloController::post(idProdotto, quantita, model) Pre: sessionCliente.getCliente().isPresent() Pre: ProdottoDao.existsById(idProdotto) Pre: quantita > 0
Postcondizione	Context: AggiungiAlCarrelloController::post(idProdotto, quantita, model) Post: CarrelloDao.addProdotto(idProdotto, quantita) == true Post: model.containsAttribute(carrelloAggiornato)
Invariante della classe	/

Nome classe	RitiroBottiglieController
Descrizione	Implementa la funzionalità di riconsegna delle bottiglie usare di un Cliente.
Metodi	+ post(Long idCliente, Integer numeroBottiglie, Model model): String
Invariante della classe	/

Nome metodo	+ post(Long idCliente, Integer numeroBottiglie, Model model): String
Descrizione	Registra il ritiro delle bottiglie usate per un Cliente.
Precondizione	Context: RitiroBottiglieController::post(idCliente, numeroBottiglie, model) Pre: sessionCliente.getCliente().isPresent()
Postcondizione	Context: RitiroBottiglieController::post(idCliente, numeroBottiglie, model) Post: BottigliaDao.ritiroBottiglie(idCliente, numeroBottiglie) == true Post: model.containsAttribute(ritiroBottiglie)
Invariante della classe	numeroBottiglie > 0

Nome classe	AccettazioneOrdineAdminController
Descrizione	Implementa la funzionalità di accettare un ordine in arrivo alla piattaforma per un Admin.
Metodi	+ post(Long idOrdine, Model model): String
Invariante della classe	/

Nome metodo	+ post(Long idOrdine, Model model): String
Descrizione	Accetta un ordine in arrivo alla piattaforma.
Precondizione	Context: AccettazioneOrdineAdminController::post(idOrdine, model) Pre: sessionAdmin.getAdmin().isPresent() Pre: OrdineDao.existsById(idOrdine) Pre: OrdineDao.findOrdineById(idOrdine).getStato() == "Da assegnare"
Postcondizione	Context: AccettazioneOrdineAdminController::post(idOrdine, model) Post: OrdineDao.accept(idOrdine) == true Post: model.Ordine.existsById(idOrdine) == true
Invariante della classe	/

Nome classe	NegazioneOrdineAdminController
Descrizione	Implementa la funzionalità di rifiutare un ordine in arrivo alla piattaforma per un Admin.
Metodi	+ post(Long idOrdine, Model model): String
Invariante della classe	/

Nome metodo	+ post(Long idOrdine, Model model): String
Descrizione	Rifiuta un ordine in arrivo alla piattaforma.
Precondizione	Context: NegazioneOrdineAdminController::post(idOrdine, model) Pre: sessionAdmin.getAdmin().isPresent() Pre: OrdineDao.existsById(idOrdine) == true Pre: OrdineDao.findOrdineById(idOrdine).getStato() == "Da assegnare"
Postcondizione	Context: NegazioneOrdineAdminController::post(idOrdine, model) Post: OrdineDao.reject(idOrdine) == true Post: model.Ordine.existsById(idOrdine) == false
Invariante della classe	/

3.6 Package Ottimizzazione Consegne Control

Nome classe	PercorsoController
Descrizione	Implementa la creazione e la visualizzazione dei percorsi per un Admin.
Metodi	+ get(Model model) : Model model + post(Map<Arco, Integer> raggruppamento) : Itinerario itinerario
Invariante della classe	/

Nome metodo	+ get(Model model) : Model model
Descrizione	Implementa la funzionalità di visualizzazione dei percorsi di un Admin.
Precondizione	Pre: sessionAdmin.getAdmin().isPresent()
Postcondizione	Context: PercorsoController::get(Model model)
Invariante della classe	/

Nome metodo	+ post(Map<Arco, Integer> raggruppamento, Model model) : Itinerario itinerario
Descrizione	Implementa la funzionalità di creazione dei percorsi.
Precondizione	Pre: sessionAdmin.getAdmin().isPresent() Pre: raggruppamento.isEmpty() == false
Postcondizione	Context: PercorsoController::post(raggruppamento, model) Post: itinerario.getList().isEmpty() == false
Invariante della classe	/

Nome classe	RaggruppamentoController
Descrizione	Implementa la creazione e la visualizzazione dei raggruppamenti per un Admin.
Metodi	+ get(Model model) : Model model + post(List<Ordine> ordini, Integer corrieri) : List<Map<Arco, Integer>> raggruppamenti
Invariante della classe	/

Nome metodo	+ get(Model model)
Descrizione	Implementa la funzionalità di visualizzazione dei raggruppamenti di un Admin.
Precondizione	Pre: sessionAdmin.getAdmin().isPresent()
Postcondizione	Context: RaggruppamentoController::post(Model model)
Invariante della classe	/

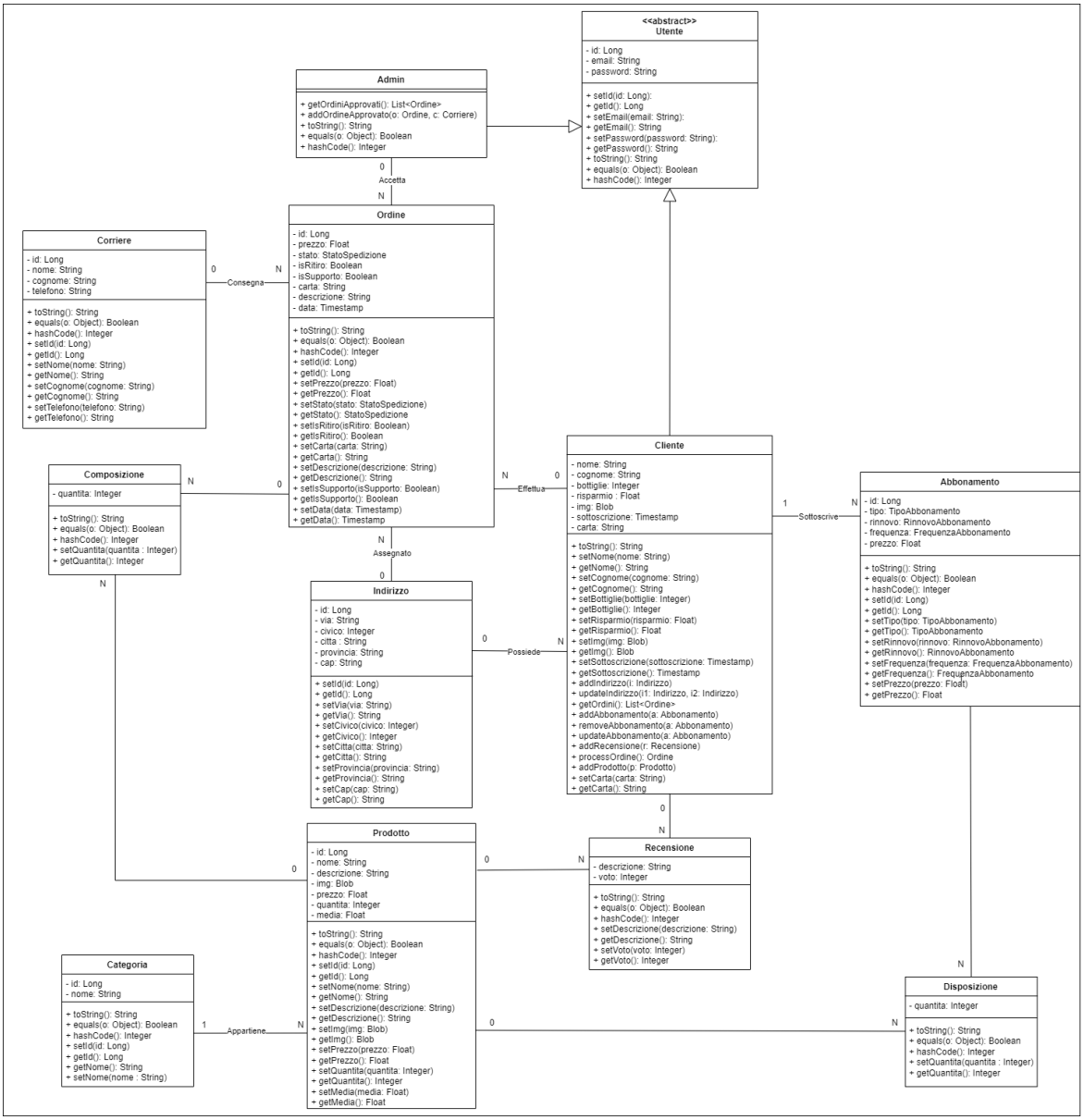
Nome metodo	+ post(List<Ordine> ordini, Integer corrieri) : List<Map<Arco, Integer>> raggruppamenti
Descrizione	Implementa la funzionalità di creazione dei raggruppamenti.
Precondizione	Pre: sessionAdmin.getAdmin().isPresent() Pre: ordini.isEmpty() == false Pre: corrieri > 0
Postcondizione	Context: RaggruppamentoController::post(ordini, corrieri)
Invariante della classe	/

Nome classe	AssegnazioneRaggruppamentoController
Descrizione	Implementa l'assegnazione dei raggruppamenti a Corrieri per un Admin.
Metodi	+ post(Map<Arco, Integer> raggruppamento, Corriere corriere)
Invariante della classe	/

Nome metodo	+ post(Map<Arco, Integer> raggruppamento, Corriere corriere)
Descrizione	
Precondizione	Pre: sessionAdmin.getAdmin().isPresent() Pre: raggruppamento.isEmpty() == false Pre: corriere != null
Postcondizione	Context: AssegnazioneRaggruppamentoController::post(ordini, corrieri) Post: OrdineDao.findByCorriere(corriere).isEmpty() == false
Invariante della classe	/

4 Class Diagram

Di seguito è riportato il Class Diagram delle Entity individuate durante la stesura del RAD: (link all'immagine)



5 Design Patterns

In questa sezione vengono descritti nel dettaglio i design pattern usati nello sviluppo della WebApp GreenBottle. Sarà fornita una breve introduzione teorica, per poi proseguire con la spiegazione di quale fosse il problema da risolvere.

5.1 Adapter

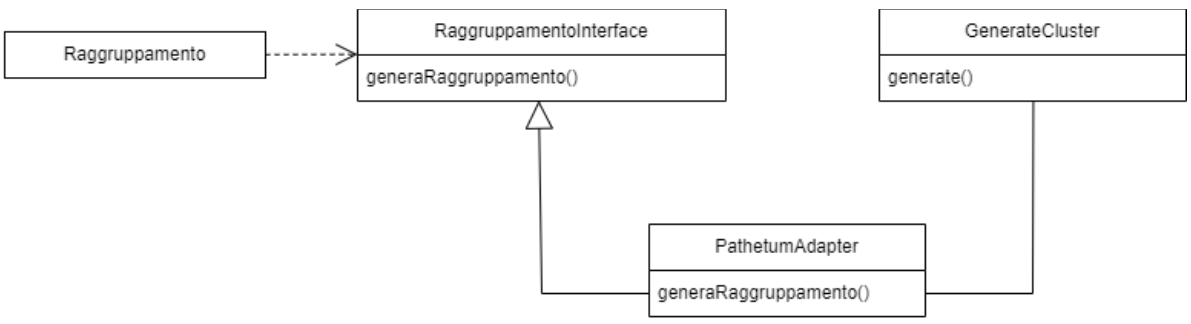
L'Adapter è un design pattern strutturale, cioè un pattern che si occupa del modo in cui le classi e gli oggetti formano strutture complesse e che riduce l'accoppiamento tra classi tramite l'incapsulamento e la creazione di classi astratte che facilitino estensioni future.

Per tali caratteristiche, l'Adapter è incredibilmente utile per quanto riguarda l'integrazione delle componenti off the shelf, che possono essere incapsulate affinché restino separate dal sistema. Il progetto godrà dunque di un elevato livello di disaccoppiamento e sarà poco impattato dalle componenti esterne.

Per implementarlo c'è bisogna di due componenti: un'interfaccia i cui metodi sono implementati in termini di richieste alla componente esterna ed una classe "adapter" che la implementa e che si occupa di delegare le richieste.

GreenBottle pone l'accento sulla sostenibilità ed uno degli aspetti caratteristici della piattaforma è la consegna delle bottiglie tramite veicoli elettrici. Per garantire che vengano effettuate nel modo più rispettoso possibile dell'ambiente è necessario, inoltre, che i corrieri seguano percorsi ottimali quando effettuano le loro consegne. A tal proposito, l'Adapter è utilizzato per il **calcolo del percorso ottimale** per i corrieri, elaborato dal modulo Pathetum esterno e visualizzato nella piattaforma web GreenBottle.

L'integrazione con il modulo di Intelligenza Artificiale viene realizzata tramite un'API che restituisce dati in formato JSON. L'Adapter converte questi dati grezzi in oggetti utilizzabili all'interno del sistema GreenBottle.

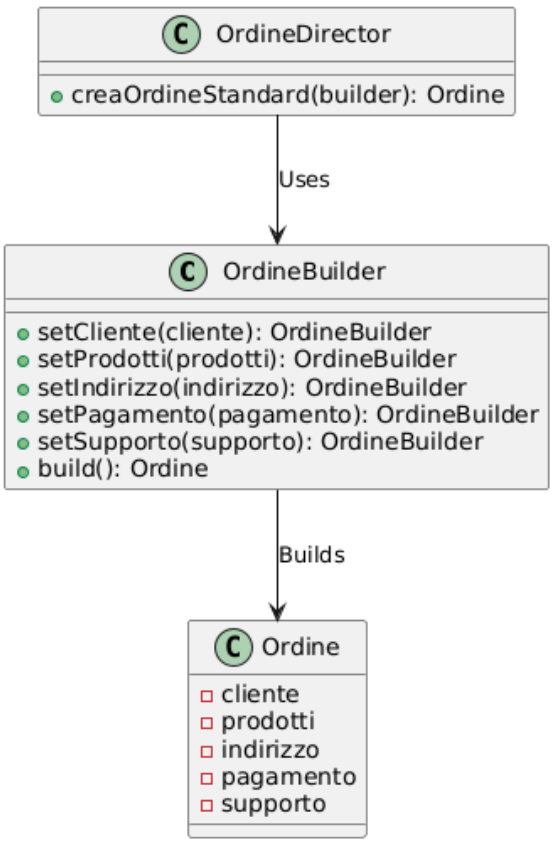


5.2 Builder Pattern

Il Builder è un pattern creazionale, cioè un pattern che fornisce un'astrazione del processo di creazione degli oggetti e aiutano a rendere un sistema indipendente dai meccanismi di creazione, composizione e rappresentazione degli oggetti.

Nel pratico, il Builder Pattern organizza la costruzione di un oggetto complesso in una serie di passi eseguiti da classi Builder. L'utilizzatore non chiama ciascuno di essi, ma soltanto quelli necessari alla creazione di una specifica configurazione di un oggetto. È possibile inoltre creare un'ulteriore classe, il Direttore, che definisce l'ordine con cui vanno intrapresi i vari passi, lasciando ai Builder il compito di implementarli. Con quest'ultima aggiunta si va ulteriormente ad incoraggiare il riuso di codice ed in più si nascondono del tutto i dettagli di creazione di un oggetto all'utente.

Il Builder Pattern rappresenta una scelta ideale per GreenBottle, data la necessità di creare oggetti complessi in formati multipli per diversi utilizzi. Questo pattern migliora la manutenibilità e la flessibilità del sistema, promuovendo un modello di oggetti chiaro e modulare. Sebbene ci siano alcuni potenziali svantaggi legati alla complessità di implementazione, i benefici in termini di decoupling e scalabilità superano ampiamente i costi, rendendolo una soluzione perfetta per indirizzi, date, ordini e profili cliente.



6 Glossario

- **Pathetum**: Sottosistema di ottimizzazione delle consegne di GreenBottle.
- **jQuery:** libreria di JavaScript per applicazioni web il cui obiettivo è quello di semplificare la selezione, la manipolazione, la gestione degli eventi e l'animazione di elementi DOM in pagine HTML, nonché semplificare le funzionalità di AJAX.
- **AJAX** (Asynchronous JavaScript and XML): tecnica di sviluppo software per la realizzazione di applicazioni web interattive, basandosi su uno scambio di informazioni in background fra client e server.
- **Bootstrap:** uno dei framework CSS più utilizzati e lo si può considerare lo standard de facto per lo sviluppo di interfacce Web.