UNIVERSITÀ DEGLI STUDI DI SALERNO

Ingegneria del Software

OBJECT DESIGN DOCUMENT

ANNO ACCADEMICO 2017/2018

Versione 1.1

Partecipanti:

NOME	MATRICOLA
Vecchione Angela	0512102274
Albano Eugenio	0512102480

Revision History:

DATA	VERSIONE	DESCRIZIONE	AUTORE
4/1/18	1.0	Stesura del documento	Membri del team.
8/1/18	1.1	Completamento del documento	Membri del team.

Indice

1.Introduzione……4

1.1 Object Design Trade-offs……..4

1.2 Linee guida per la Documentazione delle interfacce ……5

1.2.1 File Java……5

1.2.2 Naming……6

1.2.3 Uso dei commenti……6

1.3 Definizioni, acronimi e abbreviazioni……7

1.4 Riferimenti……..7

2.Packages……8

3.Interfaccia delle Classi……..9

3.1 Class Diagram……10

3.2 Descrizioni delle classi……17

3.2.2 Utente……17

3.2.3 Società Sportiva……17

3.2.4 Campo Sportivo……18

3.2.5 Carta……..18

3.2.6 Prenotazione……19

4.Glossario……19

1. INTRODUZIONE

1.1 Object Design Trade-offs

Durante la realizzazione dei documenti precedenti (RAD e SDD) abbiamo descritto a grandi linee quello che sarà il nostro sistema e quindi i nostri obiettivi, tralasciando gli aspetti implementativi. In questo documento lo scopo è di produrre un modello capace di integrare in modo coerente e preciso tutte le funzionalità individuate nelle fasi precedenti. In particolare definisce le interfacce delle classi, le operazioni, i tipi, gli argomenti e la signature dei sottosistemi definiti nel System Design. Inoltre sono specificati i trade-off e le linee guida.

Sicurezza contro Efficienza:

La sicurezza, come descritto nei requisiti non funzionali del RAD, rappresenta uno degli aspetti importanti del sistema. Tuttavia, dati i tempi di sviluppo molto limitati, ci limiteremo ad implementare sistemi di sicurezza basati su username e password degli utenti

Portabilità contro efficienza

La portabilità del sistema “Found It!” è garantita dalla scelta del linguaggio di programmazione Java. Lo svantaggio dato da questa scelta è nella perdita di efficienza introdotta dal meccanismo della macchina virtuale Java. Tale compromesso è accettabile per i numerosi supporti forniti dal linguaggio Java.

Interfaccia contro Usabilità

L’interfaccia grafica è stata realizzata in modo da essere molto semplice, chiara e concisa, fa uso di form e pulsanti disposti in maniera da rendere semplice l’utilizzo del sistema da parte dell’utente finale.

Sicurezza verso efficienza
Il sistema garantirà la buona riuscita dei pagamenti poichè effettuati attraverso pagamento con carta prepagata.
Nel nostro sistema gli Utenti (autenticati attraverso una form di login) vengono gestiti attraverso le sessioni. All'interno di ogni pagina utilizziamo delle precondizioni per gestire il controllo degli utenti, qualora l'utente digiti dal proprio browser il percorso esatto della chiamata al controller. Tali controlli sono un buon compromesso a discapito della poca efficienza persa per ogni chiamata ed aggiungono robustezza al sistema.

1.2 Linee Guida per la Documentazione delle Interfacce
Gli sviluppatori dovranno seguire alcune linee guida per la scrittura del codice.

1.2.1 File Java
Ogni file sorgente deve contenere una sola classe o interfaccia pubblica. Ogni file deve contenere nel seguente ordine:

- Dichiarazione del package
- Sezione import
- Dichiarazione di interfaccia o classe:
    - Attributi pubblici
    - Attributi privati
    - Attributi protetti
    - Costruttori
    - Altri metodi
- Classi interne

1.2.2 Naming

L'utilizzo di convenzioni sui nomi rendono il programma più leggibile e comprensibile da tutti i membri del team.

Classi e interfacce

I nomi delle classi sono nomi, composti anche da più parole, la cui iniziale è in maiuscolo. Ogni parola che compone un nome ha l'iniziale in maiuscolo.
I nomi delle classe devono essere semplici e descrittivi. Evitare l'uso di acronimi e abbreviazioni per i nomi delle classi.
E' consigliato l'uso della lingua italiana per i nomi, fatta eccezione per nomi inglesi di uso comune (esempio: TestingClass, …).

Metodi

I nomi dei metodi devono cominciare con una lettera minuscola, e le parole seguenti con la lettera maiuscola. Il nome del metodo tipicamente consiste di un verbo che identifica una azione, seguito dal nome di un oggetto.
I nomi dei metodi per l'accesso e la modifica delle variabili dovranno essere del tipo getNomeVariabile() e setNomeVariabile(). Le variabili dei metodi devono essere dichiarate appena prima del loro utilizzo e devono servire per un solo scopo, per facilitare la leggibilità.

Costanti

I nomi di costanti vengono indicati da nomi con tutte le parole in maiuscolo.
Se si tratta di un nome composto, le parole vengono separate da underscore "_".

1.2.3 Uso dei commenti

E' permesso l'utilizzo di due tipi di commenti:

- Commenti Javadoc (aree di testo compresa tra il simbolo /** e */)
- Commenti in stile C (righe delimitate da //)

L'utilizzo dei commenti Javadoc è suggerito prima della dichiarazione di:

- classi e interfacce
- costruttori
- metodi di almeno 3 righe di codice
- variabili di classe

Ogni commento, compreso tra il simbolo /** e */, deve specificare le funzionalità , la descrizione dei metodi deve apparire prima di ogni dichiarazione di metodo, e deve descriverne lo scopo.
I commenti devono essere fatti in maniera tale da rendere leggibile tale documentazione anche a sviluppatori che non posseggono l'implementazione.
I commenti di Javadoc consentono la generazione automatica della documentazione del codice, attraverso l'utilizzo di appositi tools.
I commenti stile C, ovvero le linee di codice precedute da //, sono utilizzati all'interno dei metodi, al fine di descrivere in maniera concisa e sintetica branch, cicli, condizioni o altri passi del codice.

1.3 Definizioni, acronimi e abbreviazioni

- RAD: Requirements Analysis Document
- SDD: System Design Document
- ODD: Object Design Document
- HTML: HyperText Markup LanguageJ
- JDBC: Java Database Connectivity
- JSP: Java Server Page
- MVC: Model View Controller

1.4 Riferimenti

- B. Bruegge, A. H. Dutoit, Object Oriented Software Engineering - Using UML, Pattern and Java, Prentice Hall, 3rd edition, 2009
- Guida Html, Css, JavaScript, Xml: https://www.w3schools.com/

●RAD Found It!
●SDD Found It!

2. PACKAGES

La gestione di Found It! è suddivisa in tre livelli che rappresentano la suddivisione dettata dal modello di architettura preso in considerazione per il sistema, “MVC” (Model View Controller). Ciascun livello rappresenta un package contenente le componenti relative alle funzioni associate al livello.

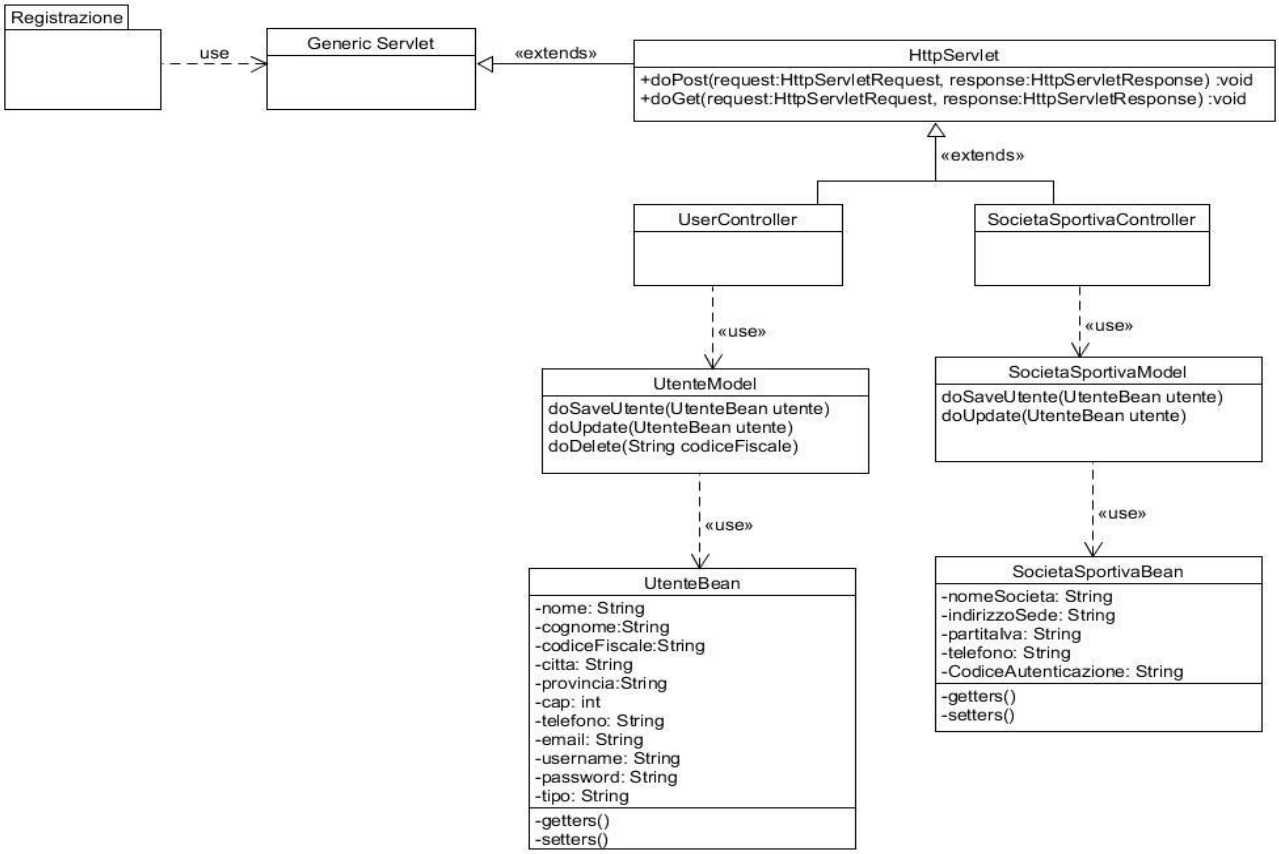
- **PACKAGE SOURCE**
  - View
    - Registrazione
    - Login
    - ProfiloUtente
    - ContattiModeratore
    - OpzioniModeratore
    - ListaCampi
    - AggiungiCampo
    - RisultatiRicercaCampi
    - Home
    - Pagamenti
    - Prenotazione
  - Controller
    - UserController
    - SocietaSportivaController
    - CampoSportivoController
    - CartaController
    - PrenotazioneController

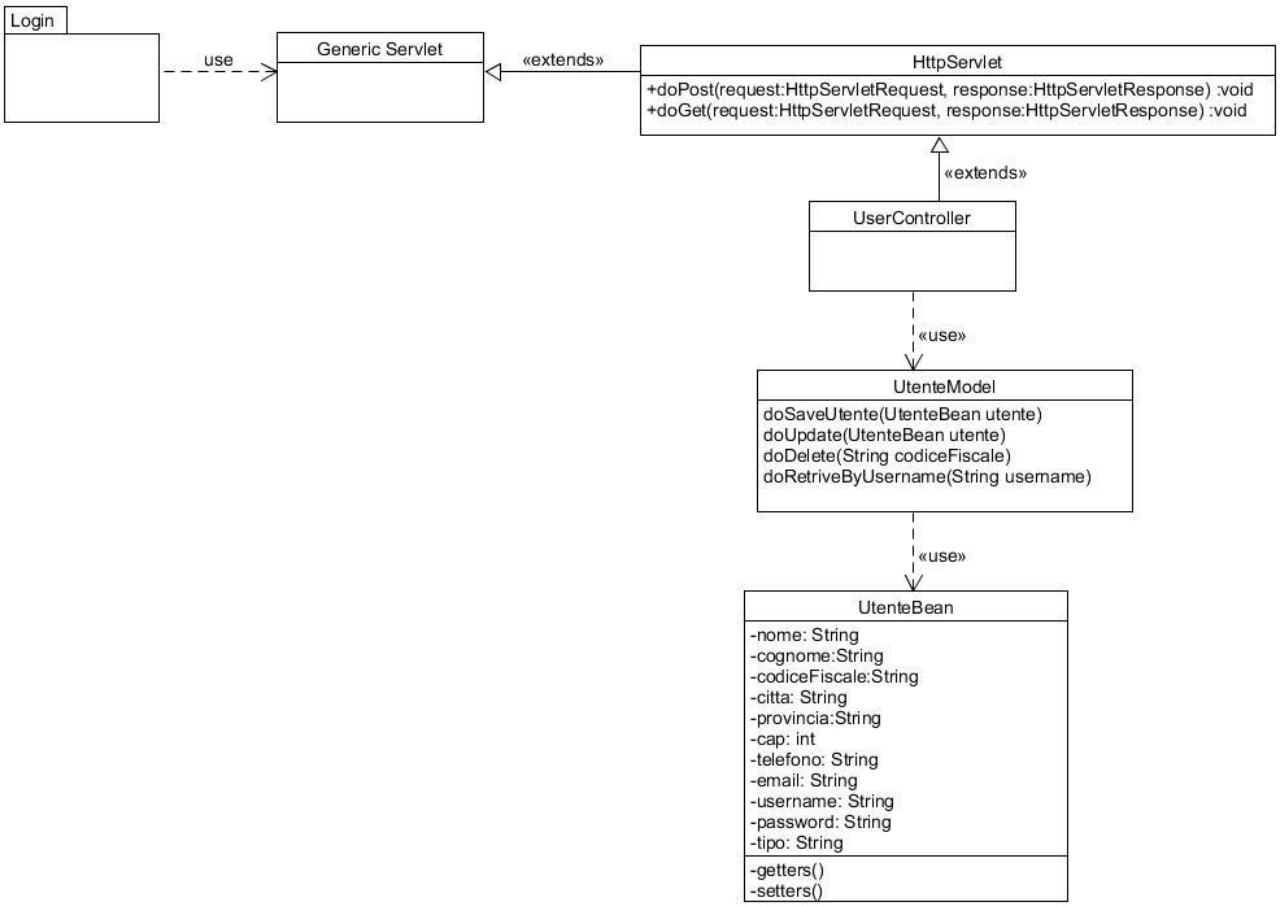
- Model
  - CampoSportivoBean
  - CampoSportivoModel
  - CartaBean
  - CartaModel
  - PrenotazioneBean
  - PrenotazioneModel
  - SocietaSportivaBean
  - SocietaSportivaModel
  - UtenteBean
  - UtenteModel

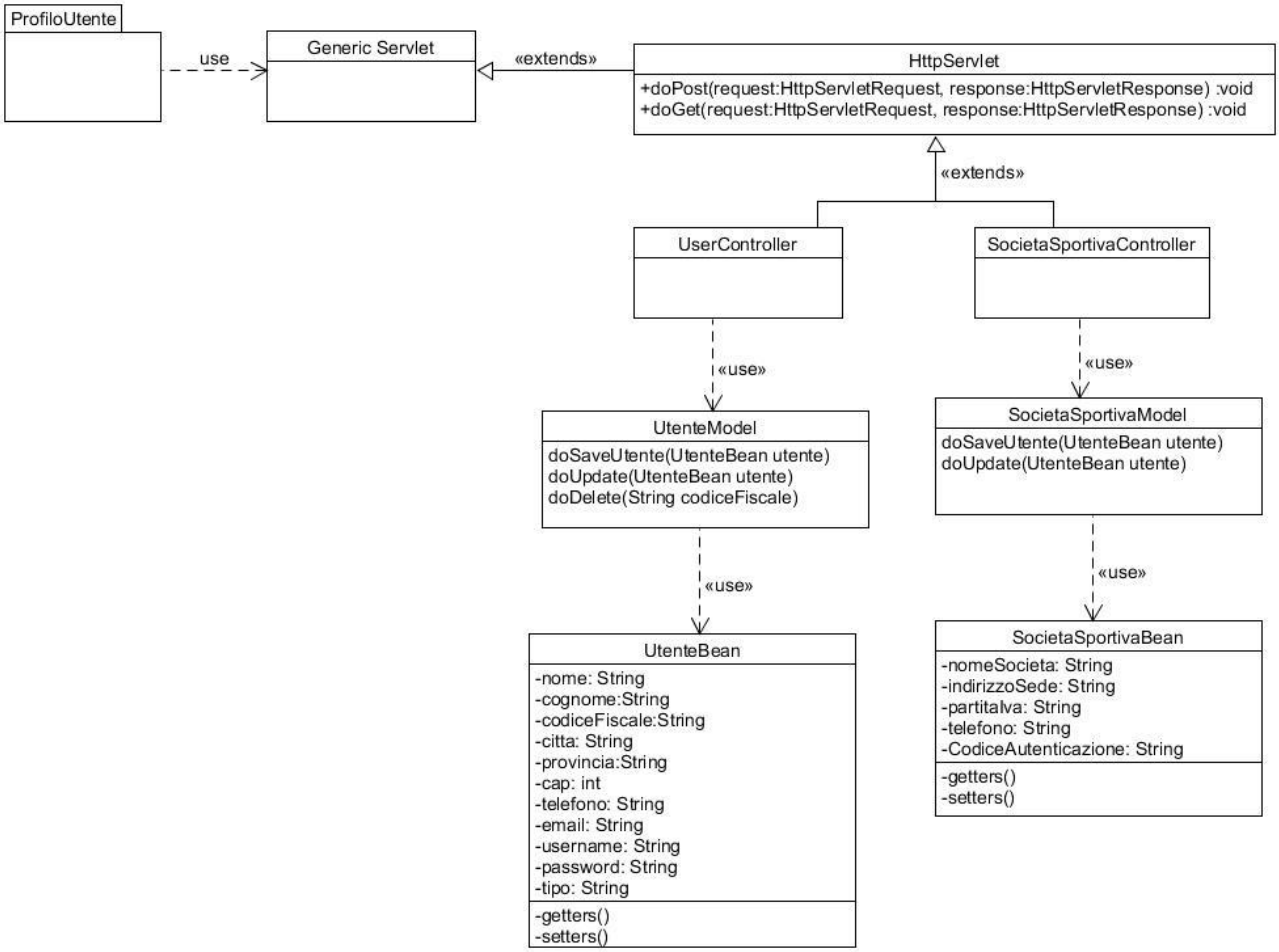
3. INTERFACCIA DELLE CLASSI

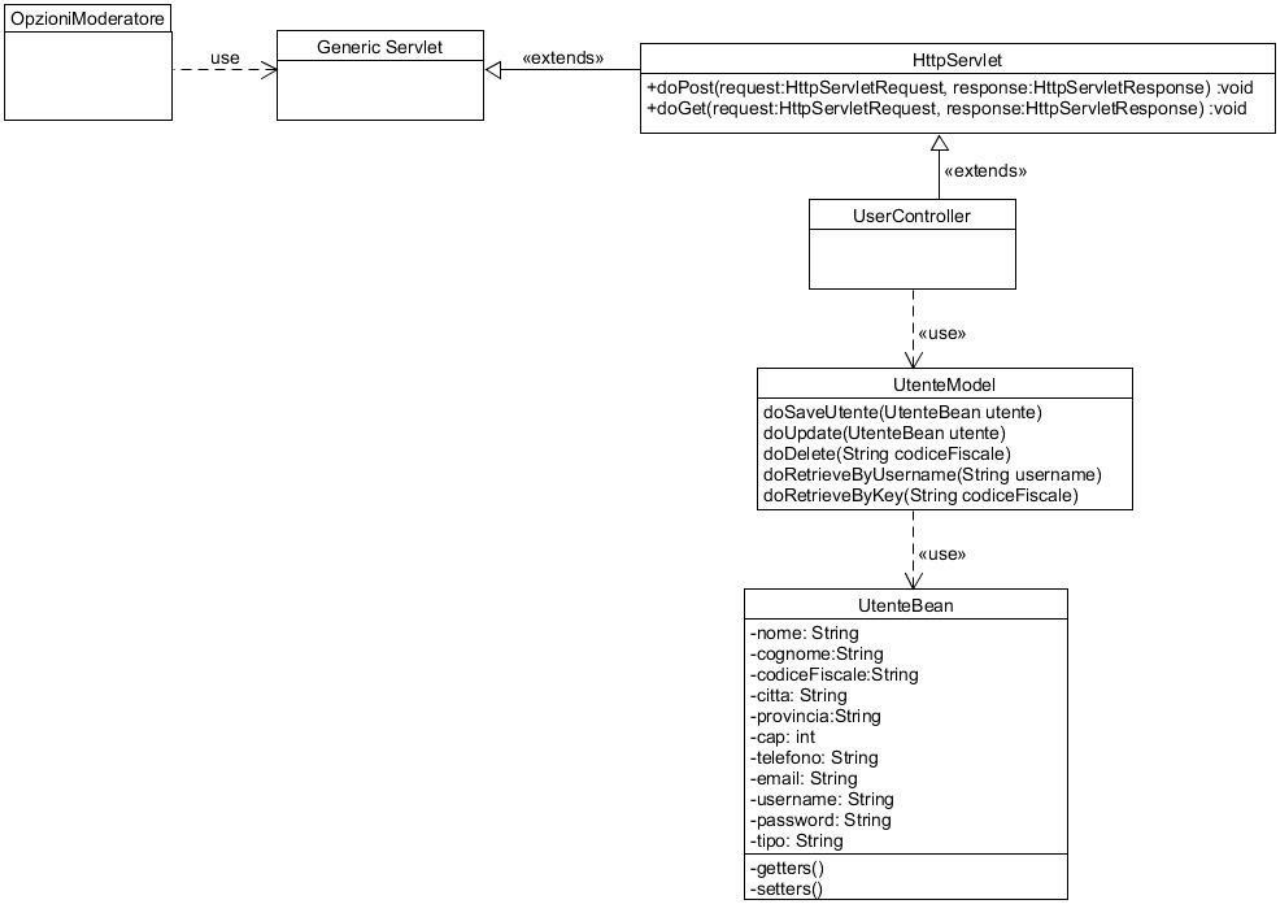
Si procede all'analisi dettagliata delle piccole classi implementate nel sistema.
L'analisi serve ad evidenziare le interfacce di interazione utilizzate nella progettazione del software.

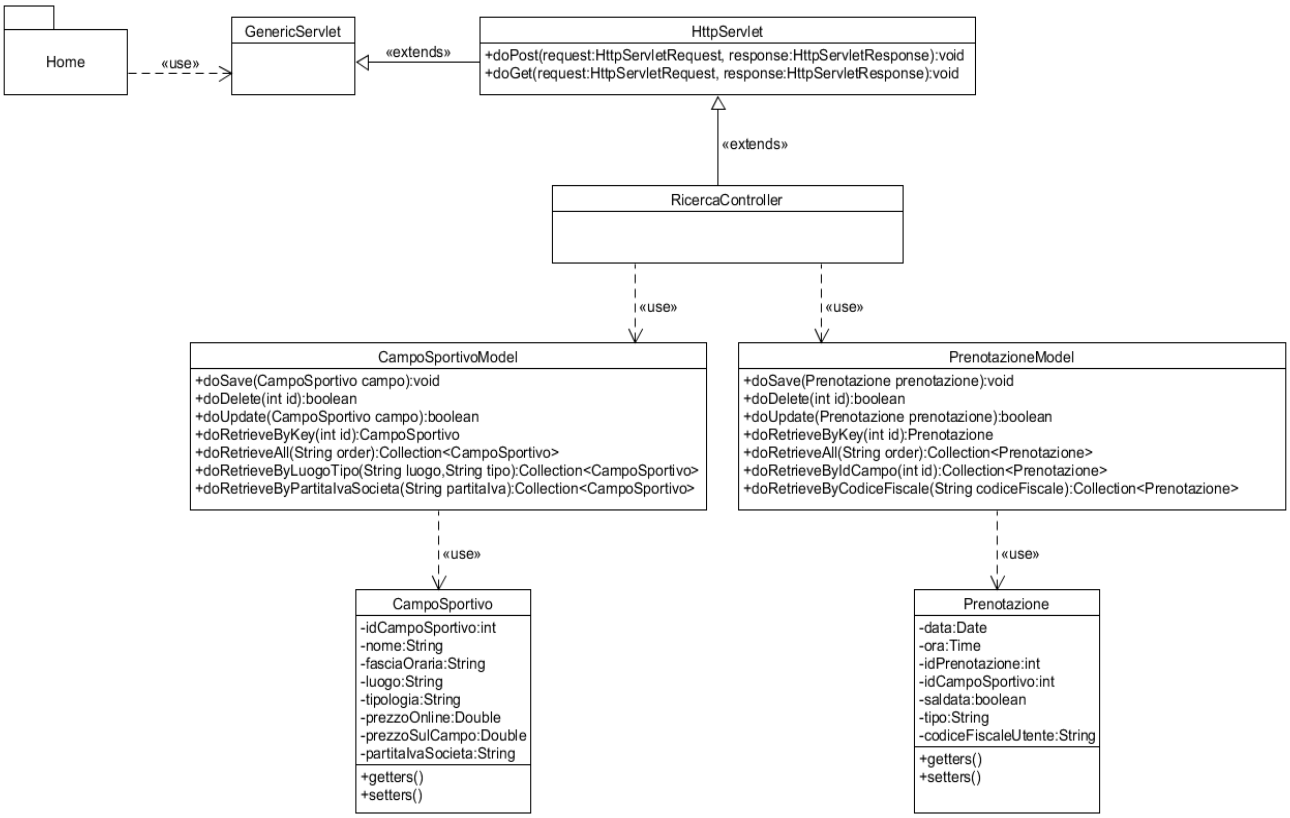
3.1 ClassDiagram

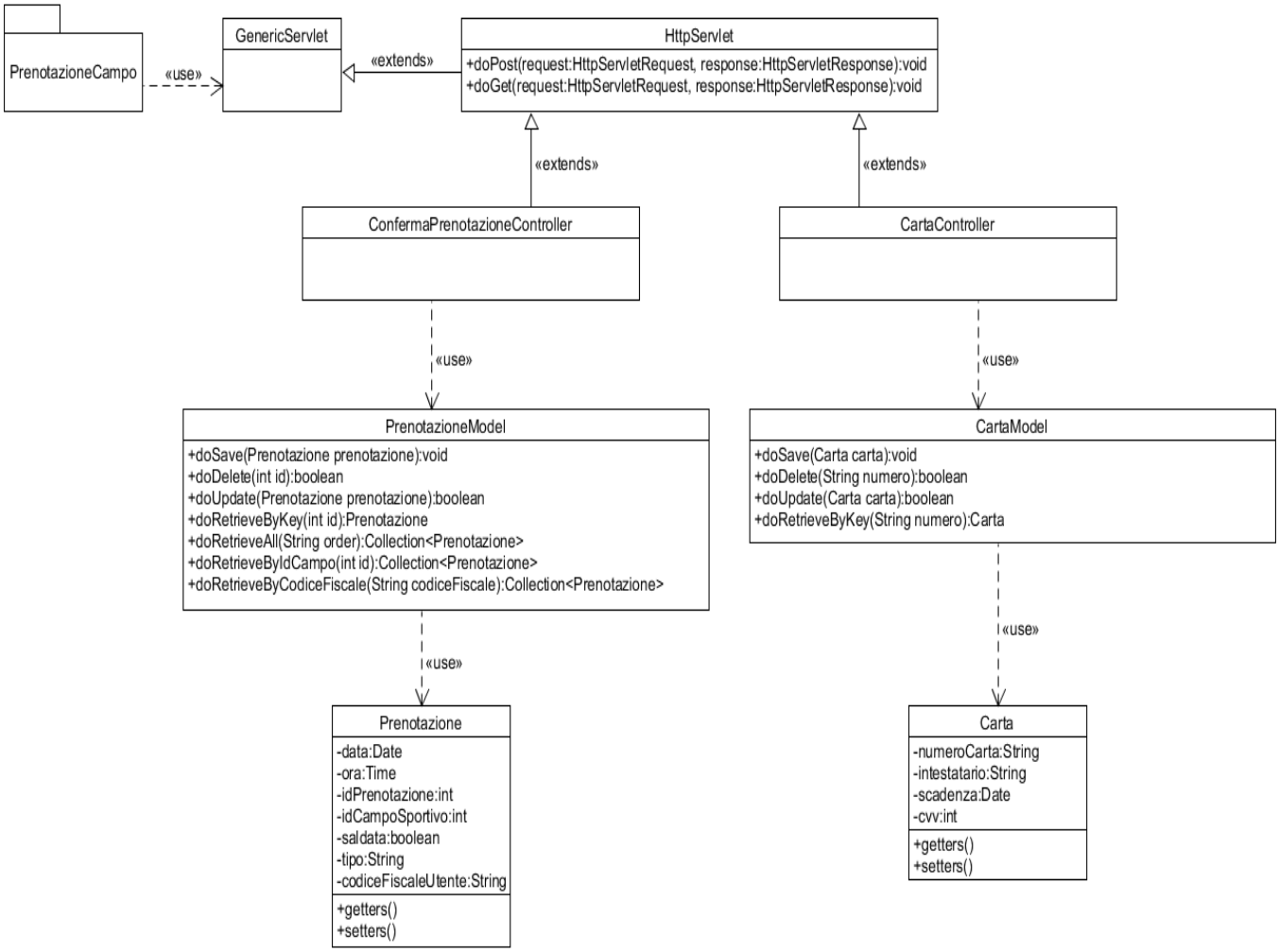


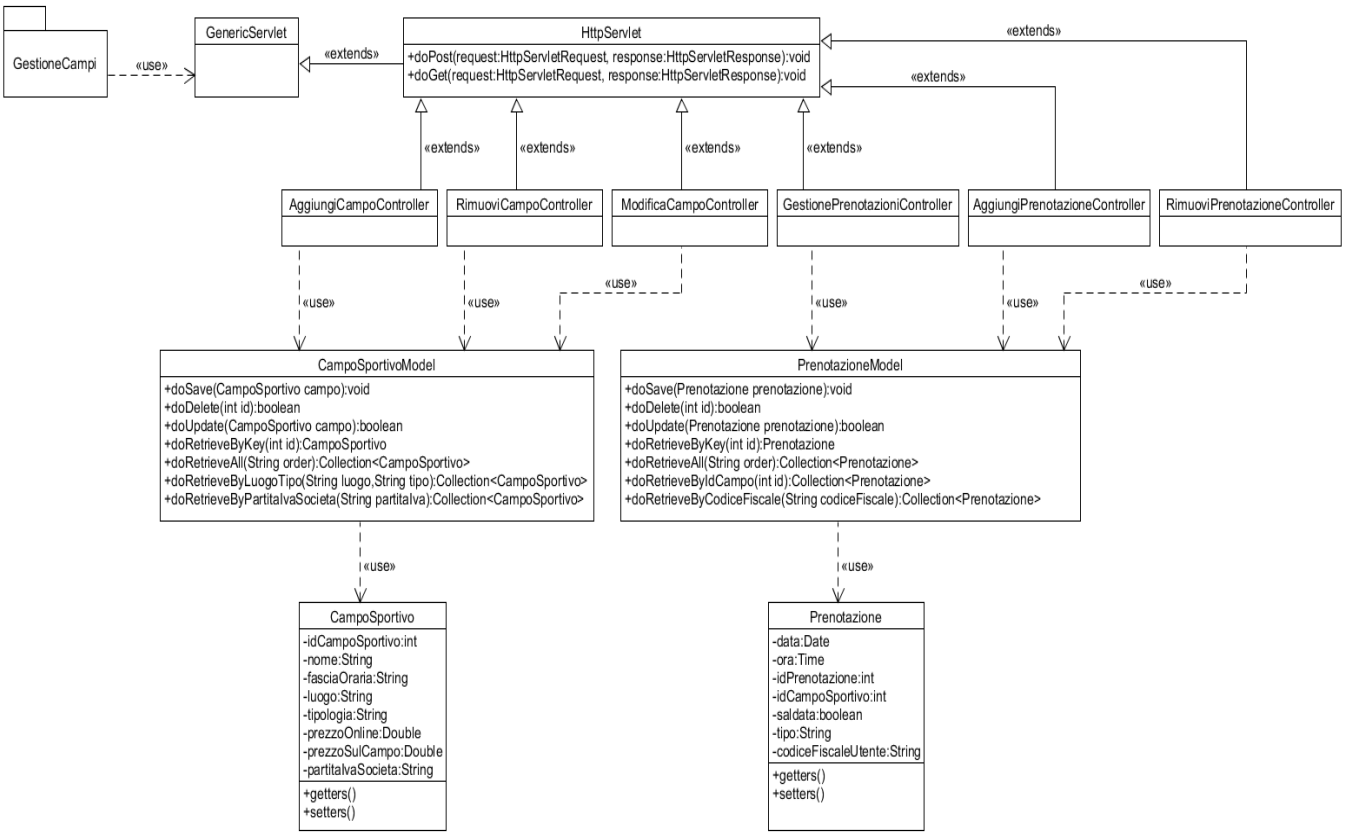

Login
use
Generic Servlet
«extends»
HttpServlet
+doPost(request:HttpServletRequest, response:HttpServletResponse) :void
+doGet(request:HttpServletRequest, response:HttpServletResponse) :void
«extends»
UserController
«use»
UtenteModel
doSaveUtente(UtenteBean utente)
doUpdate(UtenteBean utente)
doDelete(String codiceFiscale)
doRetriveByUsername(String username)
«use»
UtenteBean
-nome: String
-cognome:String
-codiceFiscale:String
-citta: String
-provincia:String
-cap: int
-telefono: String
-email: String
-username: String
-password: String
-tipo: String
-getters()
-setters()


ProfiloUtente
use
Generic Servlet
«extends»
HttpServlet
+doPost(request:HttpServletRequest, response:HttpServletResponse) :void
+doGet(request:HttpServletRequest, response:HttpServletResponse) :void
«extends»
UserController
SocietaSportivaController
«use»
«use»
UtenteModel
doSaveUtente(UtenteBean utente)
doUpdate(UtenteBean utente)
doDelete(String codiceFiscale)
SocietaSportivaModel
doSaveUtente(UtenteBean utente)
doUpdate(UtenteBean utente)
«use»
«use»
UtenteBean
-nome: String
-cognome:String
-codiceFiscale:String
-citta: String
-provincia:String
-cap: int
-telefono: String
-email: String
-username: String
-password: String
-tipo: String
-getters()
-setters()
SocietaSportivaBean
-nomeSocieta: String
-indirizzoSede: String
-partitaIva: String
-telefono: String
-CodiceAutenticazione: String
-getters()
-setters()


OpzioniModeratore
use
Generic Servlet
«extends»
HttpServlet
+doPost(request:HttpServletRequest, response:HttpServletResponse) :void
+doGet(request:HttpServletRequest, response:HttpServletResponse) :void
«extends»
UserController
«use»
UtenteModel
doSaveUtente(UtenteBean utente)
doUpdate(UtenteBean utente)
doDelete(String codiceFiscale)
doRetrieveByUsername(String username)
doRetrieveByKey(String codiceFiscale)
«use»
UtenteBean
-nome: String
-cognome:String
-codiceFiscale:String
-citta: String
-provincia:String
-cap: int
-telefono: String
-email: String
-username: String
-password: String
-tipo: String
-getters()
-setters()

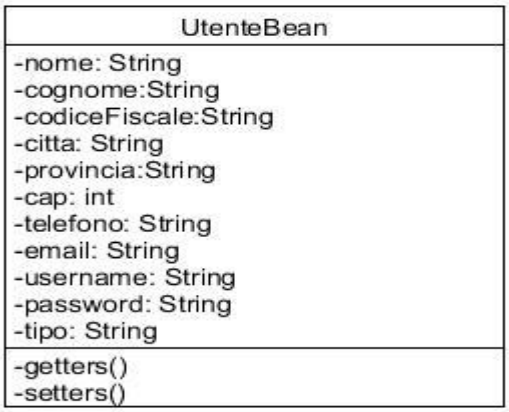

Home
«use»
GenericServlet
«extends»
HttpServlet
+doPost(request:HttpServletRequest, response:HttpServletResponse):void
+doGet(request:HttpServletRequest, response:HttpServletResponse):void
«extends»
RicercaController
«use»
«use»
CampoSportivoModel
+doSave(CampoSportivo campo):void
+doDelete(int id):boolean
+doUpdate(CampoSportivo campo):boolean
+doRetrieveByKey(int id):CampoSportivo
+doRetrieveAll(String order):Collection<CampoSportivo>
+doRetrieveByLuogoTipo(String luogo,String tipo):Collection<CampoSportivo>
+doRetrieveByPartitaIvaSocieta(String partitaIva):Collection<CampoSportivo>
PrenotazioneModel
+doSave(Prenotazione prenotazione):void
+doDelete(int id):boolean
+doUpdate(Prenotazione prenotazione):boolean
+doRetrieveByKey(int id):Prenotazione
+doRetrieveAll(String order):Collection<Prenotazione>
+doRetrieveByIdCampo(int id):Collection<Prenotazione>
+doRetrieveByCodiceFiscale(String codiceFiscale):Collection<Prenotazione>
«use»
«use»
CampoSportivo
-idCampoSportivo:int
-nome:String
-fasciaOraria:String
-luogo:String
-tipologia:String
-prezzoOnline:Double
-prezzoSulCampo:Double
-partitaIvaSocieta:String
+getters()
+setters()
Prenotazione
-data:Date
-ora:Time
-idPrenotazione:int
-idCampoSportivo:int
-saldata:boolean
-tipo:String
-codiceFiscaleUtente:String
+getters()
+setters()


PrenotazioneCampo
«use»
GenericServlet
«extends»
HttpServlet
+doPost(request:HttpServletRequest, response:HttpServletResponse):void
+doGet(request:HttpServletRequest, response:HttpServletResponse):void
«extends»
«extends»
ConfermaPrenotazioneController
CartaController
«use»
«use»
PrenotazioneModel
+doSave(Prenotazione prenotazione):void
+doDelete(int id):boolean
+doUpdate(Prenotazione prenotazione):boolean
+doRetrieveByKey(int id):Prenotazione
+doRetrieveAll(String order):Collection<Prenotazione>
+doRetrieveByIdCampo(int id):Collection<Prenotazione>
+doRetrieveByCodiceFiscale(String codiceFiscale):Collection<Prenotazione>
CartaModel
+doSave(Carta carta):void
+doDelete(String numero):boolean
+doUpdate(Carta carta):boolean
+doRetrieveByKey(String numero):Carta
«use»
«use»
Prenotazione
-data:Date
-ora:Time
-idPrenotazione:int
-idCampoSportivo:int
-saldata:boolean
-tipo:String
-codiceFiscaleUtente:String
+getters()
+setters()
Carta
-numeroCarta:String
-intestatario:String
-scadenza:Date
-cvv:int
+getters()
+setters()


GestioneCampi
«use»
GenericServlet
«extends»
HttpServlet
+doPost(request:HttpServletRequest, response:HttpServletResponse):void
+doGet(request:HttpServletRequest, response:HttpServletResponse):void
«extends»
«extends»
«extends»
«extends»
«extends»
«extends»
AggiungiCampoController
RimuoviCampoController
ModificaCampoController
GestionePrenotazioniController
AggiungiPrenotazioneController
RimuoviPrenotazioneController
«use»
«use»
«use»
«use»
«use»
«use»
CampoSportivoModel
+doSave(CampoSportivo campo):void
+doDelete(int id):boolean
+doUpdate(CampoSportivo campo):boolean
+doRetrieveByKey(int id):CampoSportivo
+doRetrieveAll(String order):Collection<CampoSportivo>
+doRetrieveByLuogoTipo(String luogo,String tipo):Collection<CampoSportivo>
+doRetrieveByPartitaIvaSocieta(String partitaIva):Collection<CampoSportivo>
PrenotazioneModel
+doSave(Prenotazione prenotazione):void
+doDelete(int id):boolean
+doUpdate(Prenotazione prenotazione):boolean
+doRetrieveByKey(int id):Prenotazione
+doRetrieveAll(String order):Collection<Prenotazione>
+doRetrieveByIdCampo(int id):Collection<Prenotazione>
+doRetrieveByCodiceFiscale(String codiceFiscale):Collection<Prenotazione>
«use»
«use»
CampoSportivo
-idCampoSportivo:int
-nome:String
-fasciaOraria:String
-luogo:String
-tipologia:String
-prezzoOnline:Double
-prezzoSulCampo:Double
-partitaIvaSocieta:String
+getters()
+setters()
Prenotazione
-data:Date
-ora:Time
-idPrenotazione:int
-idCampoSportivo:int
-saldata:boolean
-tipo:String
-codiceFiscaleUtente:String
+getters()
+setters()

3.2 Descrizioni delle classi

3.2.1 UTENTE

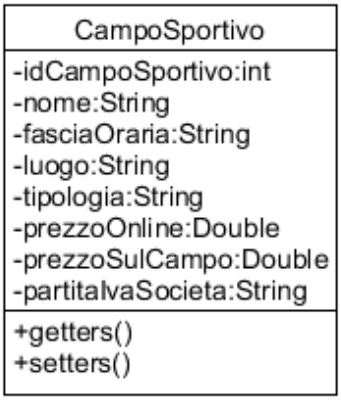


La classe Utente contiene le informazioni relative ad un generico utente con un tipo che può essere: Utente Semplice, Partner Sportivo o Moderatore. Sono inoltre presenti i metodi getters e setters della classe.

3.2.2 SOCIETA' SPORTIVA

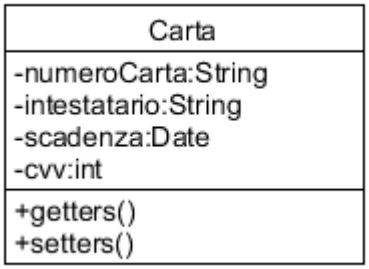
La classe Società Sportiva contiene le informazioni relative ad una Società Sportiva Sono inoltre presenti i metodi getters e setters della classe.

3.2.3 CAMPO SPORTIVO



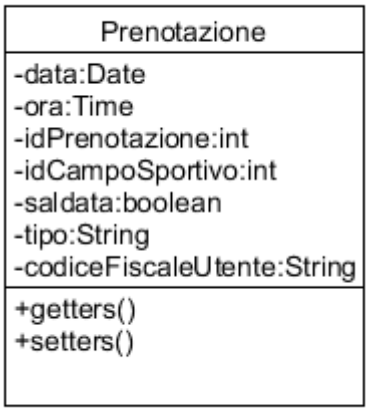
La classe Campo Sportivo contiene le informazioni relative ad un generico Campo Sportivo con una tipologia che può essere: Calcio, Calcio a 5, Basket, Pallavolo. Sono inoltre presenti i metodi getters e setters della classe.

3.2.4 CARTA



La classe Carta contiene le informazioni relative ad una generica Carta prepagata con relative informazioni per identificarla. Sono inoltre presenti i metodi getters e setters della classe.

3.2.5 PRENOTAZIONE



La classe Prenotazione contiene le informazioni relative ad una generica Carta prepagata con relative informazioni per identificarla e con un tipo che può essere: Sul Campo, Online. Sono inoltre presenti i metodi getters e setters della classe.

4. GLOSSARIO

Termini	**Descrizione**
ODD	Object Design Document
SDD	System Design Document
DB	Database management system