



UNIVERSITÀ
DEGLI STUDI
FIRENZE

Gestionale Circolo Tennis

Ingegneria del Software
2021/2022

Giovannoni Alberto
Angeli Giovanni

Indice

1 Analisi	2
1.1 Statement	2
1.2 Use case diagram	2
1.3 Use Case Template	5
2 Progettazione	7
2.1 Page navigation diagram	7
2.2 Mockup	9
2.2.1 Home-page	9
2.2.2 Menu	9
2.2.3 Login	10
2.2.4 Registrazione	11
2.2.5 Gestione	12
2.2.6 Modifica	14
2.2.7 Ticket	14
2.2.8 Contabilità	15
2.3 Class diagram	15
2.4 Database	17
2.4.1 Diagramma E-R	17
2.4.2 Schema logico e tecnologie utilizzate	17
3 Implementazione	19
3.1 Architettura	19
3.2 Classi	19
3.2.1 Model	20
3.2.2 View	20
3.2.3 Controller	20
3.2.4 DAO	20
3.2.5 JavaMail	21
3.3 Design Pattern	21
3.3.1 MVC	21
3.3.2 DAO	21
3.3.3 Façade	21
4 Testing	22

1 Analisi

1.1 Statement

Il progetto che si vuole realizzare consiste nello sviluppo di un sistema software che gestisca un circolo di Tennis composto da campi divisi per tipologia di terreno(terra rossa, cemento, sintetico) per ognuno dei quali è prevista una versione al coperto. Oltre alla prenotazione dei campi da parte degli utenti, il circolo rende disponibile dei corsi di tennis tenuti da istruttori.

Il sistema deve memorizzare l'anagrafica degli istruttori e degli utenti, i quali dovranno fare una breve registrazione per consentire allo stesso di gestire l'occupazione dei campi.

Un utente per registrarsi dovrà fornire le proprie informazioni per ricevere un ID che servirà in seguito per accedere ai servizi. Per effettuare la prenotazione l'utente dovrà fornire:

- la scelta fra campo privato o corso di tennis
- la scelta di un istruttore, se selezionato corso di tennis
- data e ora
- numero di partecipanti con i relativi ID
- numero di ore di gioco
- tipologia di terreno

Il sistema a quel punto verifica la disponibilità della prenotazione, tenendo conto di quelle già presenti. Inoltre fornirà automaticamente il resoconto della prenotazione (ticket), che comprende anche il prezzo totale.

Oltre ai vari utenti si definisce un Gestore che rappresenterà il proprietario del circolo e avrà la possibilità di:

- aggiungere/modificare le prenotazioni
- aggiungere/modificare anagrafica istruttori
- aggiungere/modificare i campi
- modificare/eliminare anagrafica utenti
- avere un prospetto settimanale dei profitti

Inoltre il Gestore potrà bloccare la prenotazione a determinati campi per permettere lo svolgimento delle lezioni con istruttore, notificando agli utenti, eventualmente prenotati, l'avvenuta cancellazione.

1.2 Use case diagram

Di seguito è riportato lo use case diagram relativo ai casi d'uso individuati sulla base dello statement. Gli stereotipi vengono utilizzati coi seguenti significati: << *include* >> indica un'operazione atomica facente parte di un'azione più generale, << *invoke* >> specifica un'azione svolta solo in determinate circostanze. Notare che, nel secondo schema (Figura 2), la generalizzazione indicata tra gestore e utente rappresenta il fatto che il gestore è anche un utente, pertanto Gestore ha tutti gli use case di un utente e in più ne ha di propri.



Figura 1: *Use case utente*

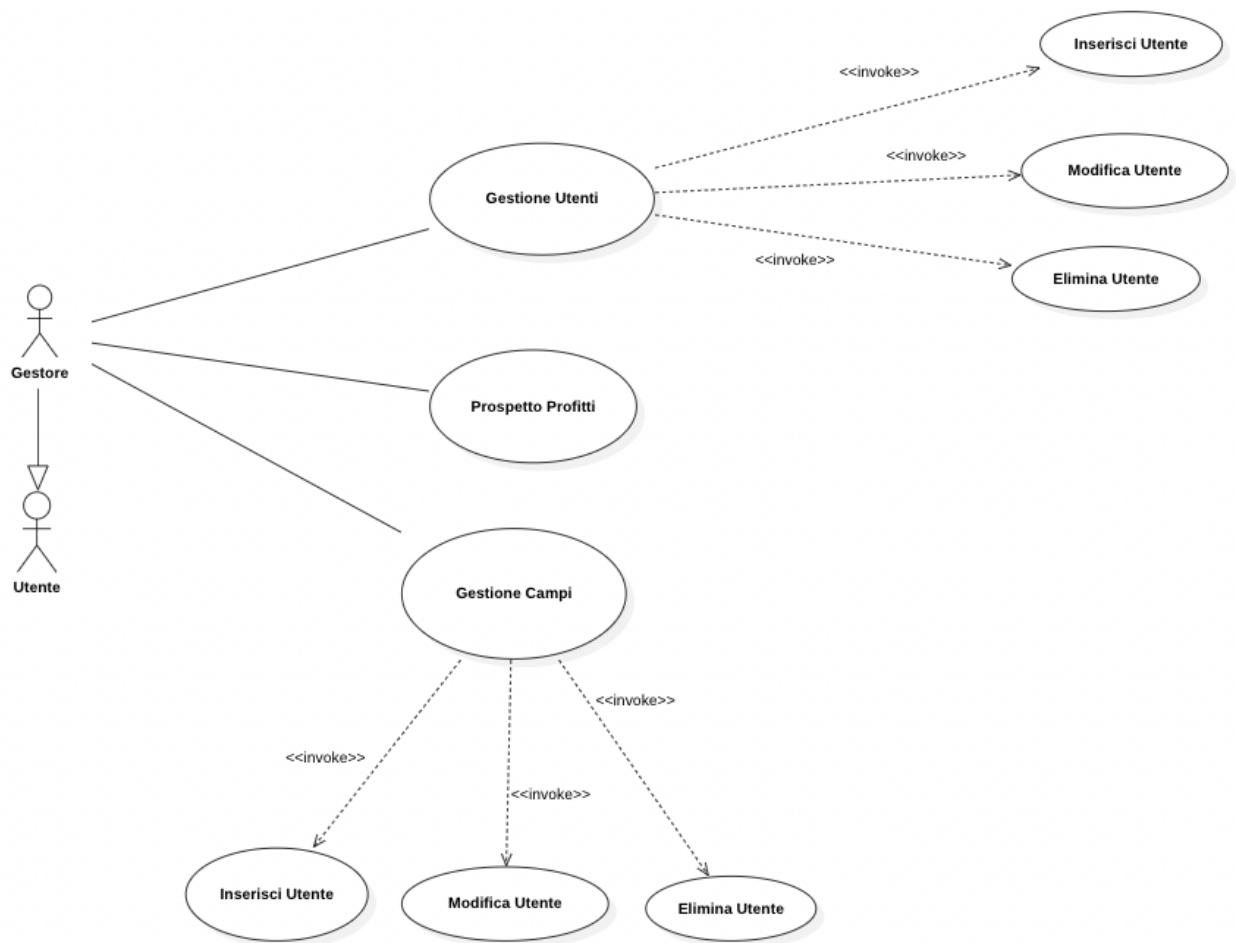


Figura 2: *Use case gestore*

1.3 Use Case Template

Vengono riportati gli use case templates atti a documentare in modo più specifico alcuni casi d'uso individuati nello use case diagram (Tabella 1 e 2). È stato scelto di riportare i templates solo per gli use case ritenuti più rilevanti ai fini del progetto.

Use Case	Registrazione
Scope	Azione riservata all'utente
Level	User Goal
Actor	Utente
Basic course	<ol style="list-style-type: none"> 1. Il nuovo utente entra pagina di registrazione (Figura 11); 2. Inserisce i propri dati personali e sceglie la password; 3. L'utente preme il pulsante "registrati"; 4. Il sistema verifica i dati inseriti; 5. I dati vengono salvati nel database; 6. L'utente riceve le proprie credenziali; 7. L'utente viene reindirizzato sulla schermata home
Alternative course	<ol style="list-style-type: none"> 4₁. Email non valida; 4₂. Password non valida; 4₃. Almeno uno dei campi è stato lasciato vuoto; 5₁. Viene segnalato un messaggio di errore

Tabella 1: *Use Case registrazione nuovo utente*

Use Case	Login
Scope	Azione di utente
Level	User Goal
Actor	Utente
Basic course	<ol style="list-style-type: none"> 1. L'utente accede alla pagina di login (Figura 8); 2. Inserisce il proprio ID e la password; 3. L'utente preme il pulsante "accedi"; 4. Il sistema verifica che ID e password corrispondano; 5. L'utente viene reindirizzato alla pagina di benvenuto;
Alternative course	<ol style="list-style-type: none"> 4₁. Il sistema è offline; 4₂. Le credenziali non trovano corrispondenza; 4₃. Almeno uno dei due campi è stato lasciato vuoto; 5₁. Viene segnalato un messaggio di errore

Tabella 2: *Use Case login dell'utente*

Use Case	Nuova prenotazione
Scope	Azione di utente
Level	Function
Actor	Utente
Basic course	<ol style="list-style-type: none"> 1. Dalla pagina di benvenuto, l'utente accede alla pagina di nuova prenotazione (Figura 10); 2. L'utente fornisce le informazioni necessarie alla prenotazione; 3. L'utente preme il pulsante "prenota"; 4. Il sistema verifica i dati inseriti; 5. La prenotazione viene salvata nel database; 6. All'utente viene mostrato il resoconto della prenotazione; 7. L'utente viene reindirizzato alla pagina di benvenuto
Alternative course	<ol style="list-style-type: none"> 4₁. Il sistema è offline; 4₂. Almeno uno dei due campi è stato lasciato vuoto; 4₃. Almeno uno dei partecipanti non è registrato; 4₄. Data e ora non disponibili; 4₅. Campo non praticabile; 4₆. Campo non disponibile per tutta la durata richiesta; 4₇. Tutti gli istruttori sono occupati; 5₁. Viene segnalato un messaggio di errore

Tabella 3: *Use Case nuova prenotazione dell'utente*

Use Case	Gestione campi
Scope	Azione riservata al gestore
Level	Function
Actor	Gestore
Basic course	<ol style="list-style-type: none"> 1. Dalla pagina di benvenuto, il gestore accede alla pagina di gestione campi (Figura 13); 2. Il gestore seleziona il campo da modificare o preme sul pulsante "+" per aggiungerne uno; 3. Il gestore fornisce i dati richiesti; 4. Il sistema verifica i dati inseriti; 5. Il campo viene inserito nel database; 6. Il gestore viene reindirizzato nella schermata di benvenuto
Alternative course	<ol style="list-style-type: none"> 4₁. Il sistema è offline; 4₂. Almeno uno dei due campi è stato lasciato vuoto; 4₃. Il formato dell'immagine caricata non è supportato; 5₁. Viene segnalato un messaggio di errore

Tabella 4: *Use Case gestione campi*

2 Progettazione

2.1 Page navigation diagram

Di seguito è riportata la navigazione tra le diverse sezioni del programma. In Figura 3 troviamo la versione per gli utenti, mentre in Figura 4 per il gestore.

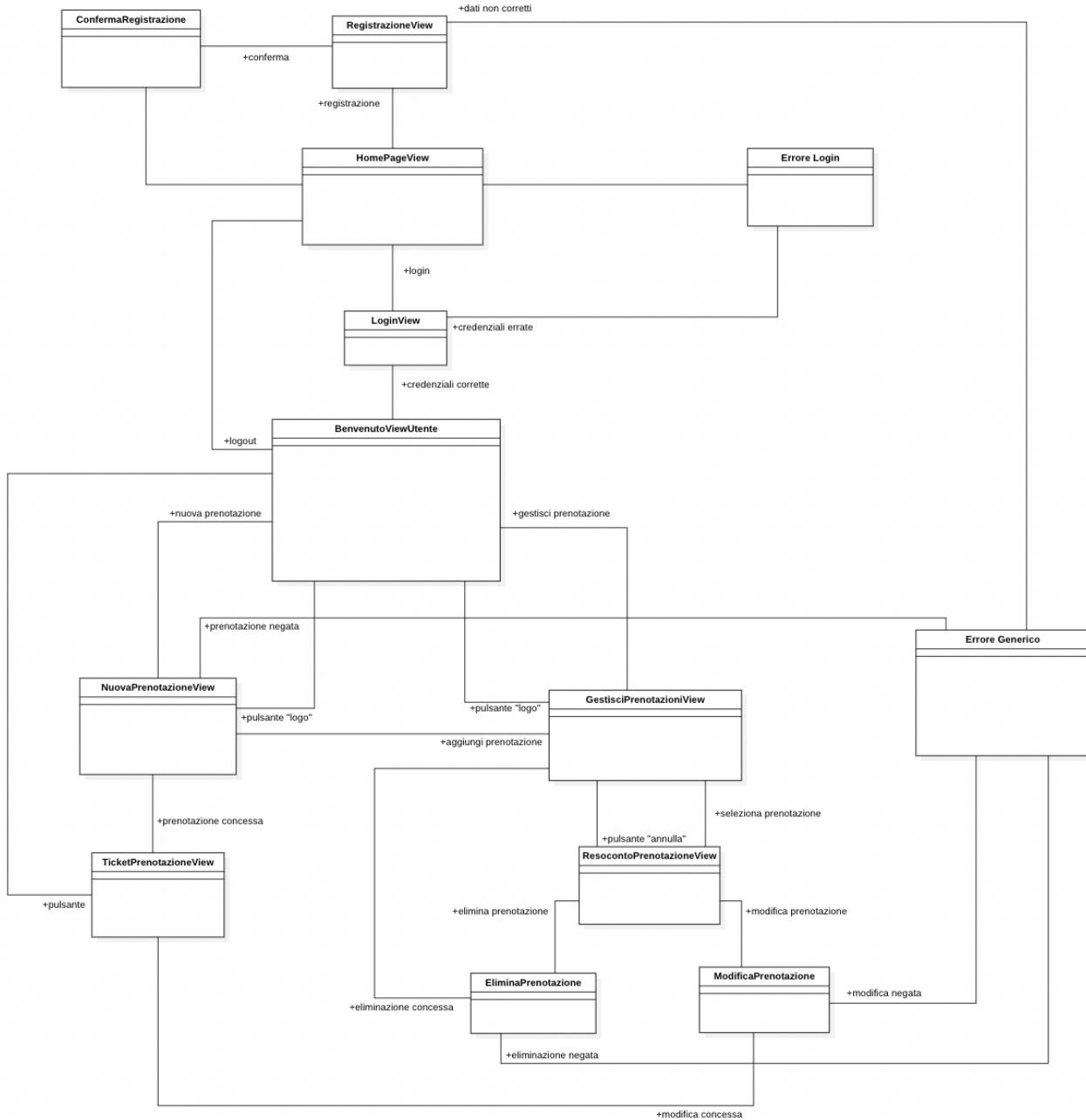


Figura 3: *Page navigation utente*

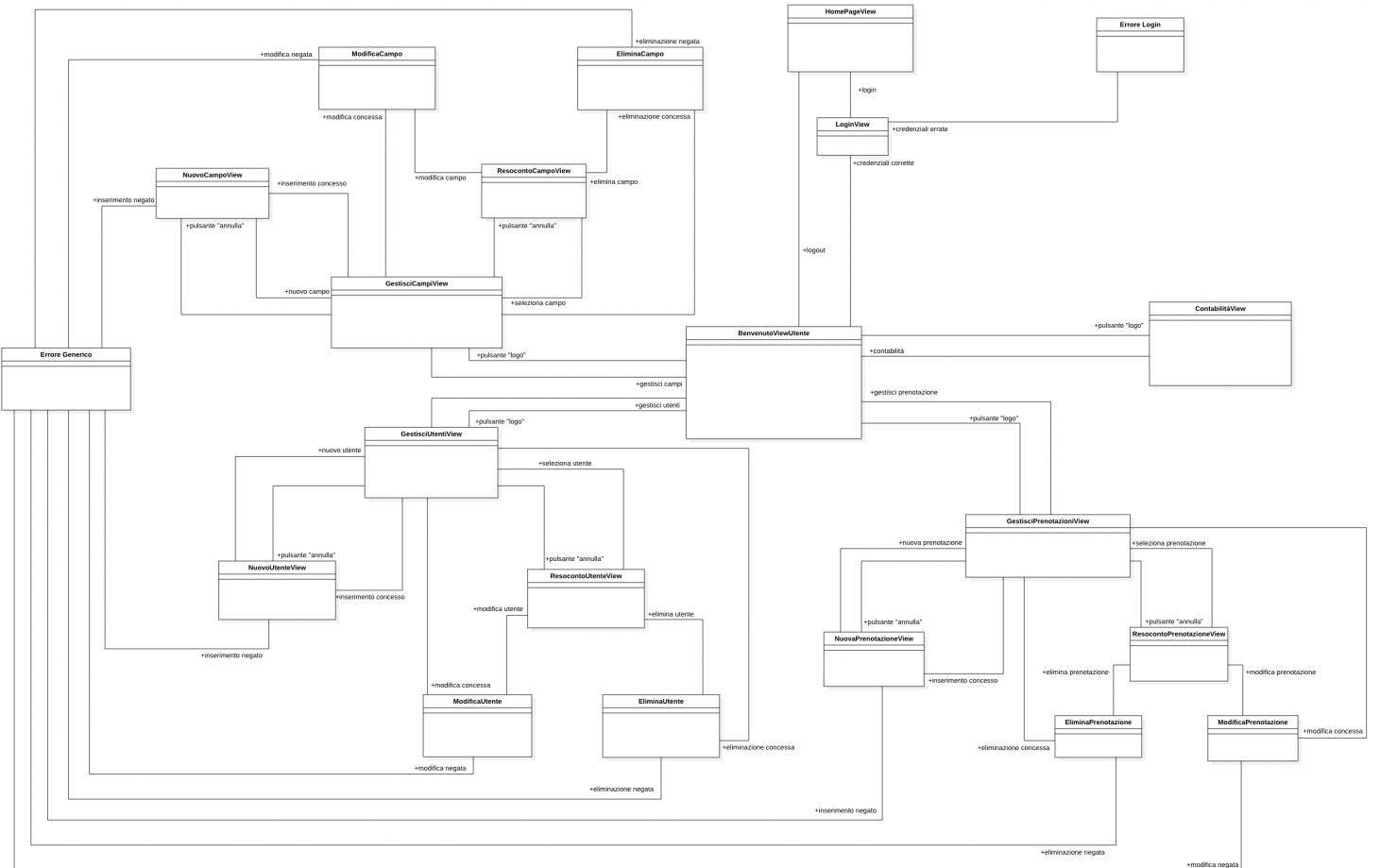


Figura 4: *Page navigation gestore*

2.2 Mockup

Di seguito si trovano delle rappresentazioni grafiche delle varie sezioni del programma, atte a mostrare come apparirebbe l'applicazione tramite un'interfaccia grafica. Tali immagini sono state realizzate su MockFlow, un tool online utilizzato per la modellazione di interfacce grafiche.

2.2.1 Home-page

Rappresenta la pagina iniziale dell'applicazione.



Figura 5: *Mockup home*

2.2.2 Menu

Schema del menu in base al tipo di client che accede.



Figura 6: *Mockup benvenuto gestore*



Figura 7: *Mockup benvenuto utente*

2.2.3 Login

Form di login che porta il client nel menu corrispondente alla sua classe (Figure 6 e 7).

A screenshot of a login form. At the top left are the same tennis ball and crest icons. The main area is light green and contains two input fields: one for "ID:" with placeholder "Inserisci ID" and another for "Password:" with placeholder "Inserisci password". Below these is a blue "Accedi" button. The entire form is enclosed in a white border.

Figura 8: *Mockup login*

2.2.4 Registrazione

Varie tipologie di form per la registrazione di un nuovo utente/campo/prenotazione.

The form is titled "nuovo campo". It includes fields for "Tipo" (with placeholder "Seleziona tipo campo"), "Prezzo" (with placeholder "Inserisci prezzo campo"), "Foto" (with a "Scegli file" button), and a checkbox for "Coperto". At the bottom are "Inserisci Campo" and "Annulla" buttons.

Figura 9: *Mockup nuovo campo*

The form is titled "nuova prenotazione". It includes fields for "Partecipanti" (with placeholder "Selezione codice partecipanti"), "Data e Ora" (with placeholder "dd/mm/yyyy" and a calendar icon), "Campo" (with placeholder "Selezione Campo"), "Durata" (with placeholder "Selezione durata"), and a checkbox for "Istruttore". At the bottom are "Prenota" and "Annulla" buttons.

Figura 10: *Mockup nuova prenotazione*

The form consists of several input fields:

- Nome:** Inserisci nome
- Cognome:** Inserisci cognome
- Email:** Inserisci email
- Password:** Inserisci password
- Età:** Inserisci età
- Sesso:** Radio buttons for M and F, with "Registrati" at the bottom.

Figura 11: Mockup registrazione

2.2.5 Gestione

Schermata che racchiude tutte le istanze di utente/campo/prenotazione.

The table displays user information:

Selezione	Codice	Nome	Cognome	Email	Età	Sesso
	I015	Janice	Monahan	Janice_Monahan@yahoo.com	34	F
	U04	Rollin	Fadel	Rollin_Fadel@gmail.com	65	M
	U190	Lera	Stroman	Lera_Stroman3@gmail.com	18	F

MODIFICA/CANCELLA UTENTE

Figura 12: Mockup gestione utenti



GESTISCI I CAMPI

SEARCH
+

Selezione	Codice	Tipo	Coperto	Valutazione	Prezzo	Foto
<input type="checkbox"/>	C93	Terra battuta	Si	★ ★ ★ ★ ★	45,00€	Campo93.jpg
<input checked="" type="checkbox"/>	C01	Sintetico	No	★ ★ ★ ★ ☆	24,00€	Campo01.jpg
<input type="checkbox"/>	C45	Erba	No	★ ★ ★ ☆ ☆	30,00€	Campo45.jpg

[MODIFICA/CANCELLA CAMPO](#)

Figura 13: *Mockup gestione campi*



GESTISCI LE TUE PRENOTAZIONI

SEARCH
+

Selezione	Codice	Partecipanti	Data e Ora	Campo	Istruttore
<input checked="" type="checkbox"/>	3910	U78, U145, U531, U413	14/01/2023 19:00-21:00	C41	I012
<input type="checkbox"/>	4031	U78	14/01/2023 19:00-21:00	C01	I230
<input checked="" type="checkbox"/>	5060	U78, U11	14/01/2023 19:00-21:00	C09	I450

[MODIFICA/CANCELLA PRENOTAZIONE](#)

Figura 14: *Mockup gestione prenotazioni*

2.2.6 Modifica

Rappresentazione della modifica/eliminazione di una prenotazione.

TENNIS
UNIVERSITÀ TECNICO STATALE DI MILANO

La tua prenotazione

Campo : C02

Codice Prenotazione : 3921

ID partecipanti : U013, U106, U258, U002

Istruttore : I105

Data e Ora : 12/10/22 17:00 - 19:00

Totale: 52 €

[Home](#)

Figura 15: Mockup modifica/eliminazione prenotazione

2.2.7 Ticket

Ticket riassuntivo generato una volta creata una nuova prenotazione.

TENNIS
UNIVERSITÀ TECNICO STATALE DI MILANO

La tua prenotazione

Campo : C02

Codice Prenotazione : 3921

ID partecipanti : U013, U106, U258, U002

Istruttore : I105

Data e Ora : 12/10/22 17:00 - 19:00

Totale: 52 €

[Home](#)

Figura 16: Mockup ticket prenotazione

2.2.8 Contabilità

Sezione dedicata alla contabilità del circolo (visionabile solo dal gestore).

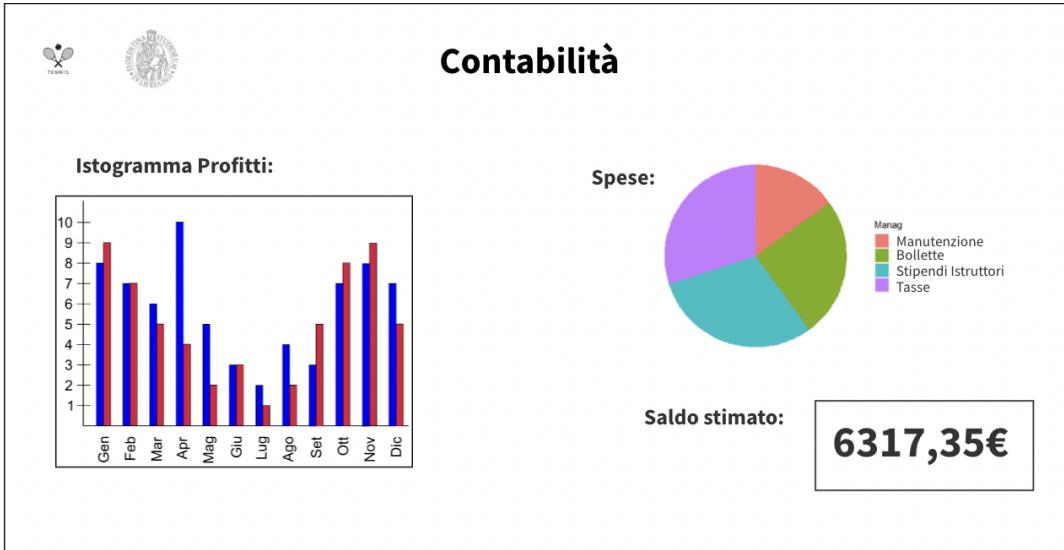


Figura 17: Mockup gestione contabilità

2.3 Class diagram

Il class diagram rappresenta le classi presenti nel programma e i metodi contenuti in esse (per semplicità non sono stati inclusi i metodi getter e setter).

Le classi sono racchiuse in 3 package principali:

- model
- view
- controller

come specificato nel pattern MVC discusso in seguito nella sezione Implementazione. All'interno del package model si trovano altri sotto-package, quali:

- campo
- persona, che contiene le classi dei diversi tipi di client
- tariffario
- prenotazione

In questo modo il codice viene suddiviso per tipologia di servizio.

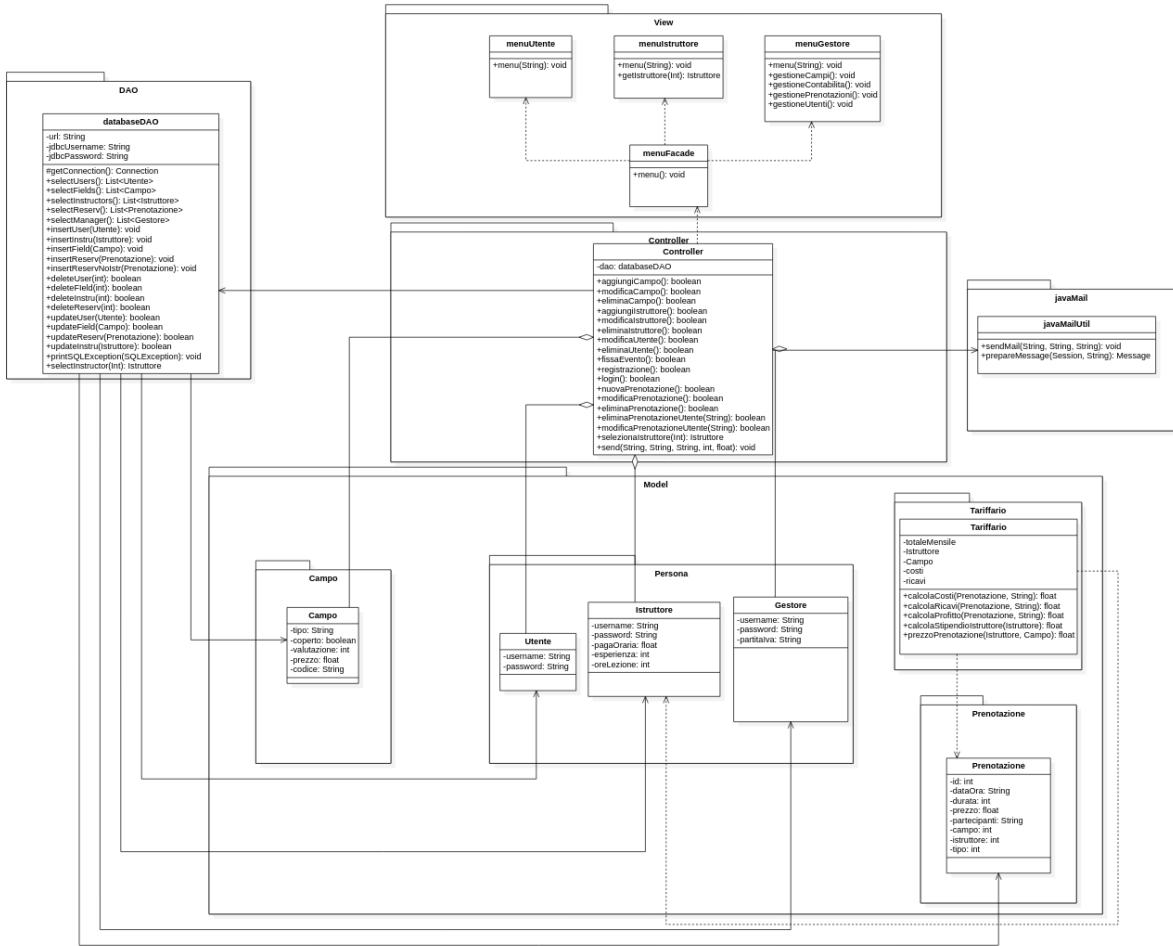


Figura 18: *Class diagram*

2.4 Database

La scelta di implementare un database è stata fatta in modo da permettere al programma di salvare i dati ottenuti da un inserimento/modifica. Questo rende questo modello di gestionale più credibile e vicino alla realtà.

2.4.1 Diagramma E-R

Il diagramma E-R serve a mostrare visivamente quali relazioni esistono fra le diverse tabelle nel database. In questo caso le tabelle *campo*, *utente* ed *istruttore* sono legate dalla relazione *prenotazione* nel seguente modo:

- in ogni prenotazione c'è uno ed un solo campo;
- l'istruttore è facoltativo, dipende dal tipo di prenotazione;
- il numero di utenti può variare da 1 a n

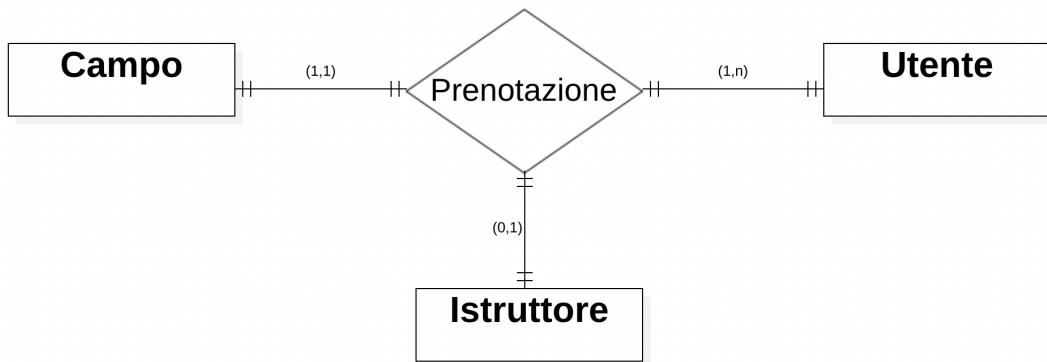


Figura 19: Schema er

2.4.2 Schema logico e tecnologie utilizzate

Il Database è stato creato con PHPMyAdmin e hostato su un server Apache locale, il tutto gestito da XAMPP. In seguito è riportato lo schema logico delle tabelle del database:

#	Nome	Tipo	Codifica caratteri	Attributi	Null	Predefinito	Commenti	Extra	Azione
1	id	int(11)		No	Nessuno		AUTO_INCREMENT	Più	
2	tipo	varchar(255)	utf8mb4_general_ci	No	Nessuno			Più	
3	prezzo	decimal(10,0)		No	Nessuno			Più	
4	valutazione	int(3)		No	Nessuno			Più	
5	coperto	tinyint(1)		No	Nessuno			Più	
6	codice	varchar(10)	utf8mb4_general_ci	No	Nessuno			Più	

Figura 20: Tabella campo

	#	Nome	Tipo	Codifica caratteri	Attributi	Null	Predefinito	Commenti	Extra	Azione
<input type="checkbox"/>	1	id 	int(11)		No	Nessuno		AUTO_INCREMENT	 Modifica  Elimina  Più	
<input type="checkbox"/>	2	nome	varchar(50)	utf8mb4_general_ci	No	Nessuno			 Modifica  Elimina  Più	
<input type="checkbox"/>	3	cognome	varchar(50)	utf8mb4_general_ci	No	Nessuno			 Modifica  Elimina  Più	
<input type="checkbox"/>	4	eta	int(11)		No	Nessuno			 Modifica  Elimina  Più	
<input type="checkbox"/>	5	email	varchar(50)	utf8mb4_general_ci	No	Nessuno			 Modifica  Elimina  Più	
<input type="checkbox"/>	6	telefono	varchar(50)	utf8mb4_general_ci	No	Nessuno			 Modifica  Elimina  Più	
<input type="checkbox"/>	7	username	varchar(50)	utf8mb4_general_ci	No	Nessuno			 Modifica  Elimina  Più	
<input type="checkbox"/>	8	password	varchar(50)	utf8mb4_general_ci	No	Nessuno			 Modifica  Elimina  Più	
<input type="checkbox"/>	9	sesso	varchar(25)	utf8mb4_general_ci	No	Nessuno			 Modifica  Elimina  Più	
<input type="checkbox"/>	10	partitalva	int(11)		No	Nessuno			 Modifica  Elimina  Più	
<input type="checkbox"/>	11	campo 	int(11)		No	Nessuno			 Modifica  Elimina  Più	

Figura 21: *Tabella gestore*

	#	Nome	Tipo	Codifica caratteri	Attributi	Null	Predefinito	Commenti	Extra	Azione
<input type="checkbox"/>	1	id 	int(11)		No	Nessuno		AUTO_INCREMENT	 Modifica  Elimina  Più	
<input type="checkbox"/>	2	nome	varchar(50)	utf8mb4_general_ci	No	Nessuno			 Modifica  Elimina  Più	
<input type="checkbox"/>	3	cognome	varchar(50)	utf8mb4_general_ci	No	Nessuno			 Modifica  Elimina  Più	
<input type="checkbox"/>	4	eta	smallint(6)		No	Nessuno			 Modifica  Elimina  Più	
<input type="checkbox"/>	5	sesso	varchar(20)	utf8mb4_general_ci	No	Nessuno			 Modifica  Elimina  Più	
<input type="checkbox"/>	6	email	varchar(20)	utf8mb4_general_ci	No	Nessuno			 Modifica  Elimina  Più	
<input type="checkbox"/>	7	telefono	varchar(20)	utf8mb4_general_ci	No	Nessuno			 Modifica  Elimina  Più	
<input type="checkbox"/>	8	username	varchar(10)	utf8mb4_general_ci	No	Nessuno			 Modifica  Elimina  Più	
<input type="checkbox"/>	9	password	varchar(20)	utf8mb4_general_ci	No	Nessuno			 Modifica  Elimina  Più	
<input type="checkbox"/>	10	esperienza	int(11)		No	Nessuno			 Modifica  Elimina  Più	
<input type="checkbox"/>	11	oreLezione	int(11)		No	Nessuno			 Modifica  Elimina  Più	
<input type="checkbox"/>	12	pagaOraria	decimal(10,0)		No	Nessuno			 Modifica  Elimina  Più	

Figura 22: *Tabella istruttore*

	#	Nome	Tipo	Codifica caratteri	Attributi	Null	Predefinito	Commenti	Extra	Azione
<input type="checkbox"/>	1	id 📃	int(11)		No	Nessuno		AUTO_INCREMENT	Modifica Elimina Più	
<input type="checkbox"/>	2	dataOra 🕒	datetime		No	Nessuno			Modifica Elimina Più	
<input type="checkbox"/>	3	durata	smallint(6)		No	Nessuno			Modifica Elimina Più	
<input type="checkbox"/>	4	prezzo	decimal(10,0)		No	Nessuno			Modifica Elimina Più	
<input type="checkbox"/>	5	partecipanti	varchar(50)	utf8mb4_general_ci	No	Nessuno			Modifica Elimina Più	
<input type="checkbox"/>	6	campo 📃	int(11)		No	Nessuno			Modifica Elimina Più	
<input type="checkbox"/>	7	istruttore 📃	int(11)		No	Nessuno			Modifica Elimina Più	
<input type="checkbox"/>	8	tipo	int(11)		No	Nessuno			Modifica Elimina Più	

Figura 23: Tabella prenotazione

	#	Nome	Tipo	Codifica caratteri	Attributi	Null	Predefinito	Commenti	Extra	Azione
<input type="checkbox"/>	1	id 📃	int(11)		No	Nessuno		AUTO_INCREMENT	Modifica Elimina Più	
<input type="checkbox"/>	2	nome	varchar(50)	utf8mb4_general_ci	No	Nessuno			Modifica Elimina Più	
<input type="checkbox"/>	3	cognome	varchar(50)	utf8mb4_general_ci	No	Nessuno			Modifica Elimina Più	
<input type="checkbox"/>	4	eta	smallint(6)		No	Nessuno			Modifica Elimina Più	
<input type="checkbox"/>	5	sesso	varchar(20)	utf8mb4_general_ci	No	Nessuno			Modifica Elimina Più	
<input type="checkbox"/>	6	email	varchar(20)	utf8mb4_general_ci	No	Nessuno			Modifica Elimina Più	
<input type="checkbox"/>	7	telefono	varchar(20)	utf8mb4_general_ci	No	Nessuno			Modifica Elimina Più	
<input type="checkbox"/>	8	username 📃	varchar(10)	utf8mb4_general_ci	No	Nessuno			Modifica Elimina Più	
<input type="checkbox"/>	9	password	varchar(20)	utf8mb4_general_ci	No	Nessuno			Modifica Elimina Più	

Figura 24: Tabella utente

3 Implementazione

3.1 Architettura

Il programma è stato implementato a partire dalla progettazione che è stata discussa precedentemente seguendo il modello MVC. I dati utili al programma sono invece salvati in un database, accessibile dal *controller* seguendo il paradigma DAO. Viene fornito anche un servizio di messaggistica tramite Email sfruttando il framework JavaMail, il quale consente di inviare Email quando necessario.

3.2 Classi

Di seguito vengono riportate e analizzate le classi più rilevanti con le loro responsabilità all'interno del programma, divise secondo il pattern MVC.

3.2.1 Model

Le classi contenute nel package Model racchiudono la logica e le caratteristiche del back-end del software. Queste sono divise in vari sotto-package al fine di costruire un sistema in cui le diverse parti comunicano ma rimangono separate.

I sotto-package che si trovano nel model sono:

- **Utente:** rappresenta un utente con la relativa anagrafica. Gli attributi *username* e *password* servono a quest'ultimo per poter effettuare il login. L'*username* inizia con la lettera "U", in questo modo il programma riconosce che ad accedere è stato un client di tipo utente.
- **Istruttore:** rappresenta un istruttore e la propria anagrafica, con informazioni, come *pagaOraria* e *oreLezione*, che servono al programma a calcolare le spese da inserire nella contabilità del gestore. Il suo *username* inizia con la lettera "I".
- **Gestore:** rappresenta il gestore del circolo, racchiude attributi relativi alla sua anagrafica. Il suo *username* inizia con la lettera "G".
- **Tariffario:** si occupa di gestire la contabilità del circolo, di calcolare il prezzo di una prenotazione, considerando il costo del campo, dell'istruttore, delle luci (se la prenotazione è in un orario successivo alle 19).
- **Prenotazione:** ha il compito di istanziare oggetti che tengono traccia delle informazioni relative alla prenotazione stessa, come *data* e *ora*, il *campo*, i *partecipanti* ecc.
- **Campo:** racchiude le informazioni del campo negli attributi *tipo* (del terreno di gioco), *coperto* (indica se il campo è al coperto o meno), *valutazione* (grado di apprezzamento del campo), *prezzo*. Il suo codice inizia con la lettera "C".

3.2.2 View

Il package view racchiude le classi che rappresentano l'interfaccia del sistema. Tali classi servono per comunicare direttamente con il client che accede al programma, con interazioni personalizzate in base al tipo di client che vi accede. Le classi sono:

- menuUtente
- menuIstruttore
- menuGestore
- menuFacade

Le prime tre sono derivate dall'ultima che, attraverso il design pattern facade, funge da interfaccia unica con i diversi client. In questo modo sarà ignaro del fatto che esistono tre tipi di oggetti (utente, istruttore, gestore). I menu interagiscono attraverso la scrittura su terminale, emulando il comportamento di un interfaccia grafica seguendo i Mockup. Di seguito al login, a seconda di che tipo di client ha effettuato l'accesso, viene creato il menu corrispondente. Il compito dei vari menu è guidare il client nelle operazioni da lui attuabili.

3.2.3 Controller

Il package controller contiene la classe controller che ha il compito di eseguire tutte le operazioni che vengono scelte dal client attraverso le interfacce del menu nel package view. Il compito del controller è quello di finalizzare le operazioni inserendo e prelevando i dati sul database, oltre a mobilità le classi del model per eseguire le operazioni necessarie al completamento dei vari task.

3.2.4 DAO

Il package DAO contiene la classe DatabaseDAO che funge da intermediario per le operazioni fra il programma ed il database.

Al suo interno si trovano le API CRUD, con le relative query SQL, i metodi e gli attributi necessari alla connessione al database e la gestione delle eccezioni SQL.

3.2.5 JavaMail

Per notificare agli utenti i vari cambiamenti all'interno del gestionale tramite un servizio email automatico abbiamo utilizzato JavaMail API, che fornisce un framework platform-independent e protocol-independent per creare applicazioni di posta e messaggistica.

Abbiamo usato questo package di Oracle all'interno della classe *javaMailUtil* e i metodi *sendMail* e *prepareMessage* per implementare le API. Per semplicità abbiamo usato 2 email esempio per l'invio delle notifiche, ma il programma è facilmente estendibile per inviare mail a tutti gli utenti iscritti al circolo e presenti del database.

3.3 Design Pattern

In quest'ultima sezione vengono elencati e descritti i design pattern utilizzati nello sviluppo e nella progettazione del programma.

3.3.1 MVC

Il design pattern MVC è un pattern architetturale finalizzato nel dividere in package le classi del programma a seconda del compito che svolgono in quest'ultimo. Viene utilizzato per separare le responsabilità dell'applicazione in 3 parti. Nel Model si trovano quelle classi che rappresentano i soggetti dell'applicazione, gli elementi contententi i dati da memorizzare ed eventualmente modificare. La View costituisce l'interfaccia con l'utente, che nel nostro caso non è stata implementata. Infine, il Controller gestisce la logica di comunicazione tra le parti. Questo permette di mantenere separato ciò che viene mostrato all'utente (l'interfaccia) dal resto della logica.

3.3.2 DAO

Il DAO (Database Access Object) è un pattern architetturale utilizzato per separare la business logic dalla logica di accesso ai dati. Il pattern funge da intermediario col database creando un maggiore livello di astrazione aumentandone la manutenibilità.

Nel nostro caso abbiamo utilizzato lo schema Gateway in cui un'unica classe contiene tutti i metodi di JDBC (accesso al database, query, ecc...) implementando così il DAO.

Le API implementate nel nostro programma sono:

- aggiunta di un elemento nella tabella;
- modifica di un elemento nella tabella;
- eliminazione di un elemento della tabella;
- stampa degli elementi della tabella

Ognuna di queste operazioni può essere eseguita sulle tabelle campo, utente, prenotazione ed istruttore, in più la stampa può essere richiamata anche sulla tabella gestore.

I metodi che si incaricano di utilizzare le API sono richiamati dal nostro controller, questo permette di non suddividere le responsabilità su più livelli dello schema MVC.

3.3.3 Façade

Il façade è un design pattern strutturale usato per encapsulare un sottosistema complesso dietro ad una semplice interfaccia, disaccoppiando l'implementazione client dal sottosistema.

Nel nostro progetto, questo pattern è usato per nascondere le tre implementazioni di menu dietro ad un'unica interfaccia *menuFacade*. In questo modo, client di diverso tipo accedono alla stessa interfaccia di menu, ma ad esempio un istruttore potrà usare solo metodi a lui accessibili.

4 Testing

In questa sezione vengono riportati i test eseguiti con il framework JUnit tramite un plug-in di Eclipse IDE.

I test sono stati svolti seguendo il metodo Black Box sulla struttura del database, in quanto è in essa che si trovano gli oggetti del programma. Ogni test viene diviso in:

- Inserimento con successo di un oggetto;
- Fallimento di un inserimento causato dalla violazione di un vincolo;
- Modifica con successo di un oggetto;
- Fallimento di una modifica causato dalla violazione di un vincolo;

In tutto sono stati quindi creati 4 test: testUtente, testIstruttore, testCampo, testPrenotazione.

All'inizio del codice dei test è presente una sezione di setup, che ha il compito di riportare il database a uno stato analogo a quello dell'ultimo test svolto.

Di seguito si trova il setup ed il codice dei vari test implementati.

```
@BeforeAll
    static void setup() throws SQLException {
        databaseDAO d = new databaseDAO();
        List<campo> campi = d.selectFields();
        for(int i = 0; i < campi.size(); i++){
            if(campi.get(i).getCodice().equals("C242")){
                assertTrue(d.deleteField(campi.get(i).getId()));
            }
        }
        List<utente> utenti = d.selectUsers();
        for(int i = 0; i < utenti.size(); i++){
            if(utenti.get(i).getUsername().equals("U750")){
                assertTrue(d.deleteUser(utenti.get(i).getId()));
            }
        }
        List<istruttore> istruttori = d.selectInstructors();
        for(int i = 0; i < istruttori.size(); i++){
            if(istruttori.get(i).getUsername().equals("I905")){
                assertTrue(d.deleteInstru(istruttori.get(i).getId()));
            }
        }
        List<prenotazione> preno = d.selectReserv();
        for(int i = 0; i < preno.size(); i++){
            if(preno.get(i).getDataOra().equals("2022-03-23 15:00:00")){
                assertTrue(d.deleteReserv(preno.get(i).getId()));
            }
        }
    }
```

Figura 25: *Setup*

```

@Test
void testCampo() throws SQLException{
    databaseDAO d = new databaseDAO();
    campo campoCorreto = new campo(0, "erba", false, 18, 3, "C242");
    campo campoSbagliato = new campo(0, "erba", false, 18, 3, "C023");
    //Test Insert
    assertTrue(d.insertField(campoCorreto));
    assertFalse(d.insertField(campoSbagliato));
    campoCorreto = new campo(9, "erba", true, 25, 3, "C103");
    campoSbagliato = new campo(9, "erba", true, 25, 3, "C023");
    //Test Update
    assertTrue(d.updateField(campoCorreto));
    assertFalse(d.updateField(campoSbagliato));
}

```

Figura 26: *Test campo*

```

@Test
void testUtente() throws SQLException {
    databaseDAO d = new databaseDAO();
    utente utentecorreto = new utente(0,"nometest", "cognometest",
        + 20, "email@test.com", "333455", ""
        + "U750", "passwordtest", 'm');
    utente utentesbagliato = new utente(1,"nome2test", "cognome2test", 23,
        + "email2@test.com", "333455", ""
        + "U001", "passwordtest2", 'f');
    //Test Insert
    assertTrue(d.insertUser(utentecorreto));
    assertFalse(d.insertUser(utesbagliato));
    //Test Update
    utentecorreto = new utente(51,"modificatest", "modificatest",
        + 20, "modifica@test.com", "333455", ""
        + "U777", "modificatest", 'm');
    utentesbagliato = new utente(51,"modifica2test", "modifica2test",
        + 23, "modifica2@test.com", "333455", ""
        + "U001", "modificatest2", 'f');
    assertTrue(d.updateUser(utentecorreto));
    assertFalse(d.insertUser(utesbagliato));
}

```

Figura 27: *Test Utente*

```

    @Test
    void testIstruttore() throws SQLException {
        databaseDAO d = new databaseDAO();
        istruttore istruttorecorretto = new istruttore(0,"istrutest",
            + "istrutest", 20, 'm', "istruttore@test.com", "333455", ""
            + "I905", "passwordistr", 4, 50, 20);
        istruttore istruttorresbagliato = new istruttore(0,"istrutest",
            + "istrutest", 20, 'm', "istruttore@test.com", "333455", ""
            + "I001", "passwordistr", 4, 50, 20);
        //Test Insert
        assertTrue(d.insertInstru(istruttorecorretto));
        assertFalse(d.insertInstru(istruttorresbagliato));
        //Test Update
        istruttorecorretto = new istruttore(7,"modificatest", "modificatest",
            + 20, 'm', "modifica@test.com", "333455", ""
            + "I531", "modifciaistr", 4, 50, 20);
        istruttorresbagliato = new istruttore(7,"modificatest2", "modificatest2",
            + 20, 'm', "modificatest2", "333455", ""
            + "I001", "modificaistr2", 4, 50, 20);
        assertTrue(d.updateInstru(istruttorecorretto));
        assertFalse(d.updateInstru(istruttorresbagliato));
    }
}

```

Figura 28: *Test Istruttore*

```

    @Test
    void testPrenotazione() throws SQLException {
        databaseDAO d = new databaseDAO();
        prenotazione prenCorretta = new prenotazione(0, "2022-03-23 15:00:00", 1, 12,
            "U001,U002", 2, 2, 1);
        prenotazione prenSbagliata = new prenotazione(0, "2022-03-23 15:00:00", 1, 12,
            "U001,U002", 2, 2000, 1);
        //Test Insert
        assertTrue(d.insertReserv(prenCorretta));
        assertFalse(d.insertReserv(prenSbagliata));
        prenCorretta = new prenotazione(134, "2022-08-16 15:00:00", 1, 12,
            "U001,U002", 2, 2, 1);
        prenSbagliata = new prenotazione(134, "2022-08-16 15:00:00",
            1298901, 12, "U001,U002", 2, 2, 1);
        //Test Update
        assertTrue(d.updateReserv(prenCorretta));
        assertFalse(d.updateReserv(prenSbagliata));
    }
}

```

Figura 29: *Test prenotazione*