



UNIVERSITA' DEGLI STUDI DI
NAPOLI FEDERICO II

Scuola Politecnica e delle Scienze di Base
Corso di Laurea in Ingegneria Informatica

Elaborato finale in **Progettazione e sviluppo di sistemi software**

Sistema gestione prenotazioni – SPORT BOOK

Autori:

Pace Mario M63000988

Nuzzolo Andrea M63001008

Santaniello Felice M63001046

Indice

Indice.....	II
Introduzione : Avvio del progetto.....	4
Capitolo 1: Processo di sviluppo.....	5
1.1 Unified Process.....	5
1.2 Tool di supporto	7
1.2.1 Trello.....	7
1.2.2 GitHub.....	8
1.3 Altre tecniche agili	9
Capitolo 2: Documenti iniziali	10
2.1 Descrizione degli obiettivi e parti interessate.....	10
2.2 Requisiti generali e di qualità.....	10
2.2.1 FURPS+	11
2.3 Vincoli	11
Capitolo 3: Specifica dei requisiti	12
3.1 Specifica informale dei requisiti.....	12
3.2 Glossario.....	13
Capitolo 4: Analisi dei requisiti	14
4.1 Casi d'uso	14
4.1.1 Tabella casi d'uso.....	14
4.1.2 Formato breve	16
4.1.3 Formato dettagliato	17
4.1.4 Diagramma dei casi d'uso.....	24
Analisi dei costi.....	25
4.2 System Domain Model.....	31
4.3 System Sequence Diagram.....	32
4.3.1 Prenota campo.....	32
4.3.2 Annulla prenotazione	33
4.3.3 Visualizza profilo.....	33
4.4 Context diagram	34

4.4.1	Context diagram.....	34
4.4.2	Context diagram con boundary	34
Capitolo 5: Documentazione di progetto		35
5.1	Architettura logica.....	35
5.2	Package Diagram	36
5.3	Class diagram.....	37
5.3.1	Client class diagram	37
5.3.2	Server class diagram	38
5.3.3	External service class diagram	38
5.4	Sequence diagram.....	39
5.4.1	Prenota campo client	39
5.4.2	Prenota campo server	40
5.4.3	Annulla prenotazione client	41
5.4.4	Annulla prenotazione server	42
5.4.5	Visualizza profilo client	43
5.4.6	Visualizza profilo server	44
5.5	Component diagram	44
5.6	Entity Relationship diagram	45
5.7	Deployment diagram	46
Capitolo 6: Implementazione		48
6.1	Strumenti e Framework a supporto utilizzati	48
6.1.1	Visual paradigm	48
6.1.2	Eclipse.....	48
6.1.3	Apache Maven	49
6.1.4	Java RMI.....	50
6.1.5	Window builder.....	50
6.1.6	Hibernate.....	50
6.2	Struttura del codice.....	51
6.3	Design pattern utilizzati.....	52
6.3.1	Singleton	52
6.3.2	Facade	53
Capitolo 7: Testing.....		56
7.1	Testing Classe Utente.....	56
7.1.1	Test successo.....	57
7.1.2	Test fallito	57
Sviluppi futuri		58

Introduzione : Avvio del progetto

Per avviare il progetto è necessario eseguire i seguenti passi:

- Scaricare i file dalla repository (<https://github.com/marpac10/ProgettoPSSS.git>)
- Aprire MySql, creare il database con username "admin" e password "root" utilizzando lo script *initDB.sql*. Inoltre, è necessario impostare le corrette timezone per MySQL.
- Aprire Eclipse IDE for Enterprise Java Developers
- Eseguire prima ServerProxy.java , poi SistemaPagamentoProxy.java, e infine eseguire il numero desiderato di interfacce utente (UtenteInterface.java).

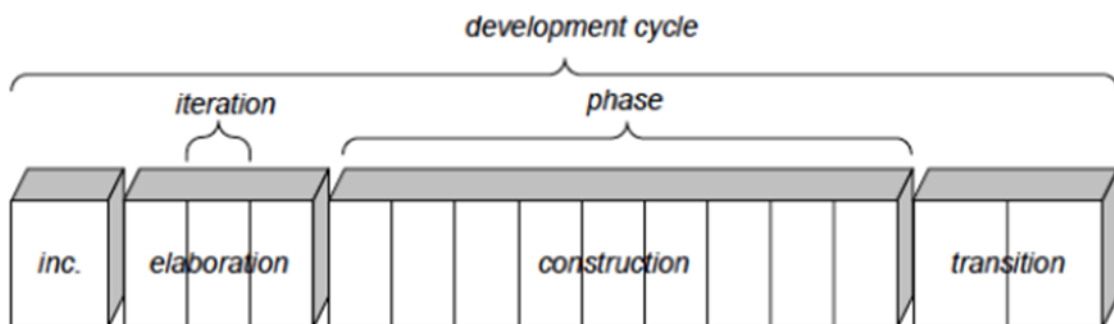
Capitolo 1: Processo di sviluppo

Per la creazione del progetto si è adottato il processo di sviluppo UP.

1.1 Unified Process

Il processo di sviluppo UP (Unified Process) è un processo iterativo evolutivo che rispecchia la proposta del modo di organizzare il lavoro per lo sviluppo software fatta dagli autori di UML. In UP non è necessario fare lo sviluppo di tutto il sistema nella sua interezza ma soltanto di una parte dei suoi requisiti che verranno elaborati in un periodo di tempo di lunghezza fissata chiamato iterazione.

Le iterazioni in genere vanno dalle 2 alle 4 settimane, e nel nostro progetto si è deciso di svolgere un'unica iterazione di 4 settimane.



Fasi del processo UP

I motivi dietro questa scelta sono numerosi:

- E' possibile applicare UML per la modellazione agile
- E' un metodo iterativo e flessibile, che permette al team di lavorare attraverso iterazioni di durata non vincolata e quindi di sentirsi meno vincolato a pratiche specifiche
- Prevede una serie di iterazioni time-boxed che possono essere viste come veri e propri mini-progetti che forniscono qualcosa di funzionante anche nel caso in cui non si riesca a terminare l'intero progetto entro una data di consegna predefinita
- Il codice è pensato per il cambiamento ed è quindi molto flessibile e riusabile

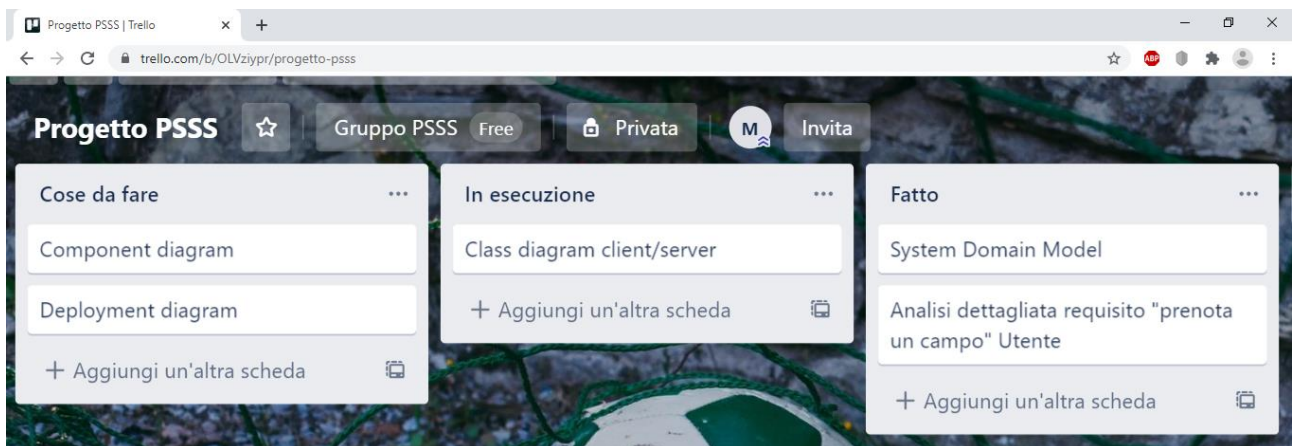
Nella nostra unica iterazione si è spesa una settimana per la fase di ideazione, e le successive tre settimane per la fase di elaborazione, nella quale sono stati sviluppati i casi d'uso che vedremo, scelti perché ritenuti essenziali per il funzionamento dell'applicazione secondo la logica prevista. Dei rimanenti requisiti sono comunque stati prodotti i modelli dei casi d'uso, seppur non ancora definitivi.

1.2 Tool di supporto

Per la fase di inception, nella quale è necessario andare a valutare gli obiettivi del progetto e la sua fattibilità, sono stati sfruttati i seguenti tool:

1.2.1 Trello

Trello è uno strumento gratuito per il project management estremamente semplice da utilizzare e particolarmente utile per lavorare in maniera agile.



Schermata di utilizzo del tool Trello

Nella schermata principale che fornisce Trello possiamo andare ad inserire le attività da fare (il To Do), quelle in esecuzione (il Doing) e quelle già completate (Done).

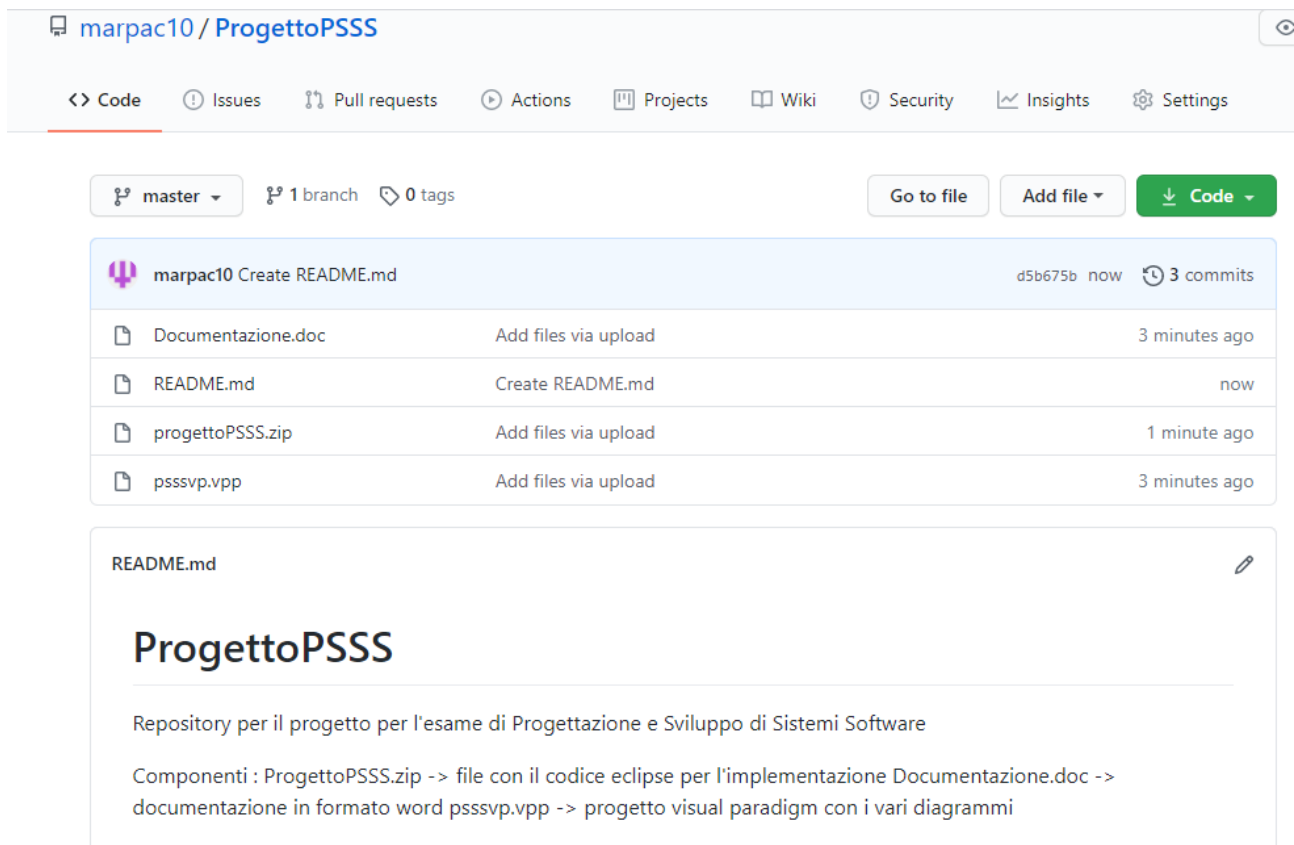
Tale strumento si è rivelato particolarmente utile per confortarsi inizialmente sulle varie idee progettuali e per tenere sotto controllo i vari artefatti in produzione durante l'iterazione.

1.2.2 GitHub

GitHub è un servizio di hosting per progetti software che è molto utilizzato dagli sviluppatori per caricare il codice sorgente dei loro programmi e renderlo scaricabile da altri utenti.

Questi ultimi possono interagire con lo sviluppatore tramite un sistema di issue tracking, pull request e commenti che permette di migliorare il codice del repository risolvendo bug o aggiungendo funzionalità. Inoltre GitHub elabora dettagliate pagine che riassumono come gli sviluppatori lavorino sulle varie versioni dei repository.

Nel nostro progetto è stato utilizzato principalmente come repository condiviso tra noi progettisti: sfruttando le operazioni di pull e push fornite da github, a turno, facevamo il commit del progetto per evitare che qualcuno di noi lavorasse su una versione vecchia.



marpac10 / ProgettoPSSS

<> Code ⓘ Issues 🔗 Pull requests ⏸ Actions 📁 Projects 📖 Wiki 🛡 Security 📈 Insights ⚙ Settings

🔗 master 1 branch 0 tags Go to file Add file Code

File	Commit Message	Commit Hash	Time
Documentazione.doc	Add files via upload	d5b675b	3 minutes ago
README.md	Create README.md		now
progettoPSSS.zip	Add files via upload		1 minute ago
psssvp.vpp	Add files via upload		3 minutes ago

README.md

ProgettoPSSS

Repository per il progetto per l'esame di Progettazione e Sviluppo di Sistemi Software

Componenti : ProgettoPSSS.zip -> file con il codice eclipse per l'implementazione Documentazione.doc -> documentazione in formato word psssvp.vpp -> progetto visual paradigm con i vari diagrammi

Al seguente link è possibile accedere al repository di progetto:

<https://github.com/marpac10/ProgettoPSSS>

1.3 Altre tecniche agili

- Daily meeting: in remoto tramite strumenti per il meeting, quali microsoft teams e skype, il team si è riunito giornalmente per lo sviluppo del progetto.
- Refactoring: continua pulizia dei cosiddetti “bad smells” del codice fatta in team.
- Continui workshop: effettuare diversi workshop e riunioni che potessero aiutare a investigare su nuovi requisiti o falle nei requisiti già specificati, sollevando eventuali polemiche su problematiche a livello implementativo o condividendo i risultati ottenuti.
- Pair programming (per quanto possibile vista la situazione covid): tecnica nella quale due programmatori lavorano insieme alla stessa postazione di lavoro. Il primo scrive il codice e l'altro svolge il ruolo di supervisore.

Capitolo 2: Documenti iniziali

In fase di avvio del progetto sono stati prodotti i seguenti documenti.

2.1 Descrizione degli obiettivi e parti interessate

Il progetto è stato pensato per facilitare il meccanismo di prenotazione di un campo sportivo da parte degli utenti. L'obiettivo principale è quello di digitalizzare la procedura, per evitare i classici problemi dovuti all'errore umano che si sono spesso verificati in quest'ambito.

I primi interessati a questo progetto sono sicuramente gli utenti, che con un semplice click dall'applicazione potranno gestire tutto il sistema di prenotazione, senza dimenticare gli utenti della "vecchia scuola", che potranno ancora prenotare con la classica telefonata al gestore del campo, che provvederà tramite la propria interfaccia a digitalizzare la prenotazione.

Altra parte interessata è ovviamente il gestore del campo, che non dovrà più impazzire tra chiamate continue, prenotazioni da effettuare e da annullare, ma avrà un'applicazione che farà tutto al posto suo, compresi i pagamenti nei casi in cui gli utenti decidano di utilizzare la propria carta di credito.

2.2 Requisiti generali e di qualità

E' richiesto un sistema molto robusto, che sappia gestire eventuali input scorretti da parte dell'utente e che mantenga ben disaccoppiati i vari layer, in modo tale da poter effettuare modifiche nel corso del tempo senza che il sistema ne risenta troppo. Inoltre deve essere un sistema ad elevate prestazioni, in modo tale da permettere agli utenti di raggiungere il loro risultato nel minor tempo possibile. E' ovviamente necessaria un'alta usabilità: le interfacce dell'applicazione dovranno essere particolarmente user-friendly per permettere una buona esperienza di utilizzo.

Legato a quest'ultimo requisito c'è quello della comprensibilità, necessaria per gli stessi motivi enunciati precedentemente.

E' possibile riassumere i seguenti requisiti (e le soluzioni adottate per realizzarli) utilizzando uno dei più noti modelli per la definizione di requisiti di qualità del software: il modello FURPS+.

2.2.1 FURPS+

FURPS+ è un modello che indica di riportare nei requisiti quelli funzionali, quelli di usabilità, cioè quando deve essere usabile un'applicazione e quali manuali e tutorial devono essere previsti se l'usabilità è un punto delicato dell'applicazione, la disponibilità e la capacità di recupero, la precisione e la correttezza dei risultati, le prestazioni e i tempi di risposta e la supportabilità, cioè supporto all'evolubilità del software, indispensabile per i sistemi che devono avere lunga durata. I requisiti del progetto possono essere così riassunti :

- F -> Functional : Necessaria la funzionalità del logging per tenere traccia del comportamento del sistema e per renderlo più sicuro, permettendo soltanto agli utenti autenticati di potervi accedere e di operare modifiche ai propri dati.
E' inoltre necessario gestire correttamente i possibili errori negli input utente, rendendo robusto il sistema ed impedendo blocchi irreparabili.
- U -> Usability : Il sistema deve possedere un'interfaccia user-friendly adatta a qualsiasi tipo di cliente, e i colori utilizzati per gli output devono essere pochi e ben marcati per evitare problemi associati al daltonismo di eventuali clienti.
- R -> Reliability : Il sistema deve essere affidabile e disponibile per tutta la durata del periodo di apertura del centro sportivo.
- P -> Performance : Il sistema deve avere alte performance e permettere in breve tempo a qualsiasi cliente di completare la propria prenotazione.
- S -> Supportability : L'applicazione deve essere adattabile per qualsiasi campo sportivo, in maniera tale da poter allargare il numero di campi sportivi gestiti senza troppe modifiche.
- + -> Ulteriori requisiti : Utilizzo di JAVA come linguaggio di programmazione.

2.3 Vincoli

L'applicazione deve essere scritta in linguaggio JAVA, un linguaggio di programmazione ad alto livello orientato agli oggetti e a tipizzazione statica, che si appoggia sull'omonima piattaforma software di esecuzione, poiché esso è stato specificatamente progettato per essere il più possibile indipendente dalla piattaforma hardware di esecuzione.

Capitolo 3: Specifica dei requisiti

Nella fase di specifica dei requisiti sono stati prodotti i seguenti artefatti.

3.1 Specifica informale dei requisiti

Il primo passo da effettuare nella fase di inception è quello di procedere con la definizione informale dei requisiti del progetto, almeno di quelli fondamentali, a mezzo di un cosiddetto *workshop*. Quest'ultimo ha quindi come output un primo documento informale di specifica che viene prodotto dall'analista, che verrà utilizzato per la formalizzazione dei primi casi d'uso.

Si vuole realizzare un'applicazione Java per la gestione di un centro sportivo, al fine di semplificare il sistema di prenotazione dei campi disponibili. Ogni campo presente nel centro sportivo è caratterizzato da un codice univoco, dalla tipologia di sport praticato, dalla tipologia e dal prezzo di prenotazione. All'avvio del server, il sistema aggiorna la lista di prenotazioni disponibili per i vari campi per la giornata odierna, eliminando tutte le prenotazioni dei giorni precedenti. Il gestore deve poter modificare la lista, aggiungendo o eliminando determinate prenotazioni. L'utente, previa autenticazione, deve poter prenotare un campo scegliendo la data e ora della prenotazione. In risposta, il sistema restituisce i campi disponibili. L'utente seleziona il campo desiderato e il sistema registra la prenotazione se il pagamento è andato a buon fine. L'utente deve anche poter scegliere di pagare al campo, in caso non possa completare il pagamento con carta di credito. In alternativa, l'utente può contattare il gestore telefonicamente riferendogli i dati della prenotazione e quest'ultimo deve poter registrare la prenotazione dalla propria interfaccia. Ogni volta che una prenotazione va a buon fine, il sistema assegna all'utente un totale di punti premium (10 punti premium in questa prima implementazione), e al raggiungimento di una soglia di punti premium (100 punti premium) l'utente passa da STANDARD a PREMIUM. Il gestore deve poter visualizzare la lista delle prenotazioni in una specifica data e ora. Un utente deve poter annullare la prenotazione e ricevere un ticket per il rimborso, o in alternativa può contattare telefonicamente il gestore che deve quindi poter annullare la prenotazione. L'utente deve poter visualizzare il proprio profilo, nel quale sono riportate la lista delle prenotazioni effettuate, i punti premium accumulati, lo stato (STANDARD o PREMIUM), gli eventi al quale è iscritto e i ticket accumulati da riscuotere. Il gestore deve poter creare un evento sportivo limitato agli utenti premium. Al momento della creazione dell'evento, un sistema di mail esterno provvede a notificare gli utenti iscritti. Gli utenti premium devono potersi iscrivere agli eventi creati dai gestori.

3.2 Glossario

CENTRO SPORTIVO	Insieme di uno o più spazi di attività sportiva dello stesso tipo o di tipo diverso.
CAMPO	Detto anche rettangolo di gioco, struttura destinata ad ospitare le partite. Può essere di più tipologie a seconda della presenza o meno della copertura e del manto erboso.
PRENOTAZIONE	L'atto di riservare un campo per una determinata fascia oraria ed un determinato giorno.
GESTORE	Soggetto che amministra i servizi offerti dal centro sportivo e manutiene le attrezzature.
UTENTE	Colui che usufruisce dei servizi di base offerti dal centro sportivo.
UTENTE PREMIUM	Colui che usufruisce dei servizi di base offerti dal centro sportivo ed altri aggiuntivi derivanti da un numero prestabilito di prenotazioni effettuate.
EVENTO	Partita organizzata dal gestore e alla quale possono iscriversi, per la partecipazione, solo utenti premium.
TICKET	Ricevuta erogata al momento dell'annullamento di una prenotazione per la quale si era già pagata la quota di partecipazione.

Capitolo 4: Analisi dei requisiti

Questa fase ha come input la documentazione informale dei requisiti e come obiettivo quello di produrre una documentazione che sia del tutto indipendente dall'implementazione.

4.1 Casi d'uso

4.1.1 Tabella casi d'uso

Utente	<p>Prenota un campo</p> <p>Annulla prenotazione</p> <p>Visualizza profilo</p> <p>Stampa ticket</p> <p>Registrazione</p> <p>Login</p> <p>Logout</p>
--------	--

Utente Premium	<p>Prenota un campo</p> <p>Annulla prenotazione</p> <p>Visualizza profilo</p> <p>Iscriviti ad evento</p> <p>Stampa Ticket</p> <p>Registrazione</p> <p>Login</p> <p>Logout</p>
Gestore	<p>Prenota un campo</p> <p>Annulla prenotazione</p> <p>Visualizza lista prenotazioni</p> <p>Modifica lista prenotazioni</p> <p>Crea Evento</p>
Amministratore	<p>Avvia server</p>

4.1.2 Formato breve

Caso d'uso	Attore	Descrizione
Prenota un campo	Utente / Utente premium	L'utente deve poter prenotare un campo tra quelli disponibili nel sistema.
Annulla prenotazione	Utente / Utente premium	L'utente deve poter annullare una prenotazione già effettuata, ricevendo un ticket per il rimborso del pagamento se effettuato.
Visualizza profilo	Utente / Utente premium	L'utente deve poter visualizzare il suo profilo, dove sono presenti prenotazioni effettuate, numero di punti premium, stato dell'utente (STANDARD/PREMIUM) e ticket accumulati.
Stampa ticket	Utente / Utente premium	L'utente deve poter stampare un ticket presente nel suo profilo al fine di consegnarlo al gestore del campo per ottenere il rimborso effettivo.
Registrazione	Utente / Utente premium	L'utente deve potersi registrare al sistema inserendo username e password desiderati.
Login	Utente / Utente premium	L'utente deve potersi loggare al sistema inserendo username e password corretti.
Logout	Utente / Utente Premium	L'utente deve poter effettuare il logout dal sistema.
Iscriviti ad evento	Utente premium	L'utente premium deve potersi iscrivere a un evento creato dal gestore.
Prenota un campo	Gestore	Il gestore deve poter prenotare un campo su richiesta dell'utente.
Annulla prenotazione	Gestore	Il gestore deve poter annullare una prenotazione su richiesta dell'utente.
Visualizza lista prenotazioni	Gestore	Il gestore deve poter visualizzare la lista di tutte le prenotazioni effettuate.
Modifica lista prenotazioni	Gestore	Il gestore deve poter modificare la lista delle prenotazioni disponibili, aggiungendole, eliminandole o modificandole.
Crea evento	Gestore	Il gestore deve poter creare eventi riservati.
Avvia server	Amministratore	L'amministratore avvia il server.

4.1.3 Formato dettagliato

Nome caso d'uso	Prenota un campo
Portata	Applicazione desktop
Livello	Obiettivo Utente
Attore primario	Utente
Parti interessate	Utente (vuole prenotare un campo), Gestore (vuole che la prenotazione vada a buon fine), Servizio di pagamento
Pre-condizioni	L'utente deve essere loggato
Garanzia di successo	La prenotazione viene salvata; La lista delle prenotazioni viene aggiornata; I dati dell'utente vengono aggiornati
Scenario principale	<ol style="list-style-type: none"> 1. L'utente sceglie di prenotare un campo 2. Il sistema chiede di scegliere il campo da prenotare 3. L'utente seleziona il campo richiesto 4. Il sistema elabora la richiesta e mostra la lista delle prenotazioni disponibili per quel campo 5. L'utente seleziona la prenotazione desiderata 6. Il sistema mostra i dettagli della prenotazione 7. L'utente effettua il pagamento 8. Il sistema aggiorna la lista delle prenotazioni 9. Il sistema assegna all'utente i punti premium
Estensioni	<p>7.a: Pagamento non andato a buon fine</p> <ol style="list-style-type: none"> 1. Il sistema segnala l'errore 2. L'utente torna alla schermata di pagamento <p>7.b: L'utente sceglie di pagare al campo</p> <ol style="list-style-type: none"> 1. Il sistema registra che il campo non è stato pagato <p>9.a: L'utente diventa premium</p> <ol style="list-style-type: none"> 1. Il sistema notifica l'utente dell'avvenuta promozione

Nome caso d'uso	Annulla prenotazione
Portata	Applicazione desktop
Livello	Obiettivo utente
Attore primario	Utente
Parti interessate	Utente (vuole annullare la prenotazione), Gestore (vuole sapere che la prenotazione è stata annullata e se dovrà essere rimborsata)
Pre-condizioni	L'utente deve essere loggato
Garanzia di successo	La prenotazione viene annullata; La lista delle prenotazioni viene aggiornata; Il ticket, se necessario, viene erogato; I dati dell'utente vengono aggiornati
Scenario principale	<ol style="list-style-type: none"> 1. L'utente sceglie di annullare una prenotazione 2. Il sistema mostra le prenotazioni effettuate 3. L'utente seleziona la prenotazione da annullare 4. Il sistema elabora la richiesta e aggiorna la lista delle prenotazioni
Estensioni	<p>4.a: L'utente aveva già pagato</p> <ol style="list-style-type: none"> 1. Il sistema mostra i dettagli del rimborso 2. L'utente conferma 3. Il sistema salva il ticket associato all'utente

Nome caso d'uso	Visualizza profilo
Portata	Applicazione desktop
Livello	Obiettivo utente
Attore primario	Utente
Parti interessate	Utente (vuole visualizzare il suo profilo)
Pre-condizioni	L'utente deve essere loggato
Garanzia di successo	La lista delle prenotazioni effettuate viene mostrata; Il numero di punti premium accumulati viene mostrato; Lo stato dell'utente (STANDARD o PREMIUM) viene mostrato; I ticket accumulati dall'utente vengono mostrati
Scenario principale	<ol style="list-style-type: none"> 1. L'utente sceglie di visualizzare il profilo 2. Il sistema mostra la lista delle prenotazioni effettuate 3. Il sistema mostra il numero di punti premium accumulati 4. Il sistema mostra lo stato dell'utente 5. Il sistema mostra i ticket accumulati dall'utente
Estensioni	

Nome caso d'uso	Registrazione
Portata	Applicazione desktop
Livello	Sotto-funzione
Attore primario	Utente
Parti interessate	Utente (vuole registrarsi al sistema), Gestore (vuole che l'utente si registri al sistema in modo che possa guadagnare sulla prenotazione dei campi)
Pre-condizioni	
Garanzia di successo	L'utente completa la registrazione; Viene assegnato un id univoco all'utente; La lista degli utenti viene aggiornata;
Scenario principale	<ol style="list-style-type: none"> 1. L'utente sceglie di registrarsi al sistema 2. L'utente inserisce username e password desiderati 3. Il sistema elabora la richiesta e memorizza username e password 4. Il sistema comunica un messaggio di avvenuta registrazione
Estensioni	<p>3.a: L'username è già presente nel sistema</p> <ol style="list-style-type: none"> 1. Il sistema segnala l'errore 2. L'utente torna alla schermata di registrazione

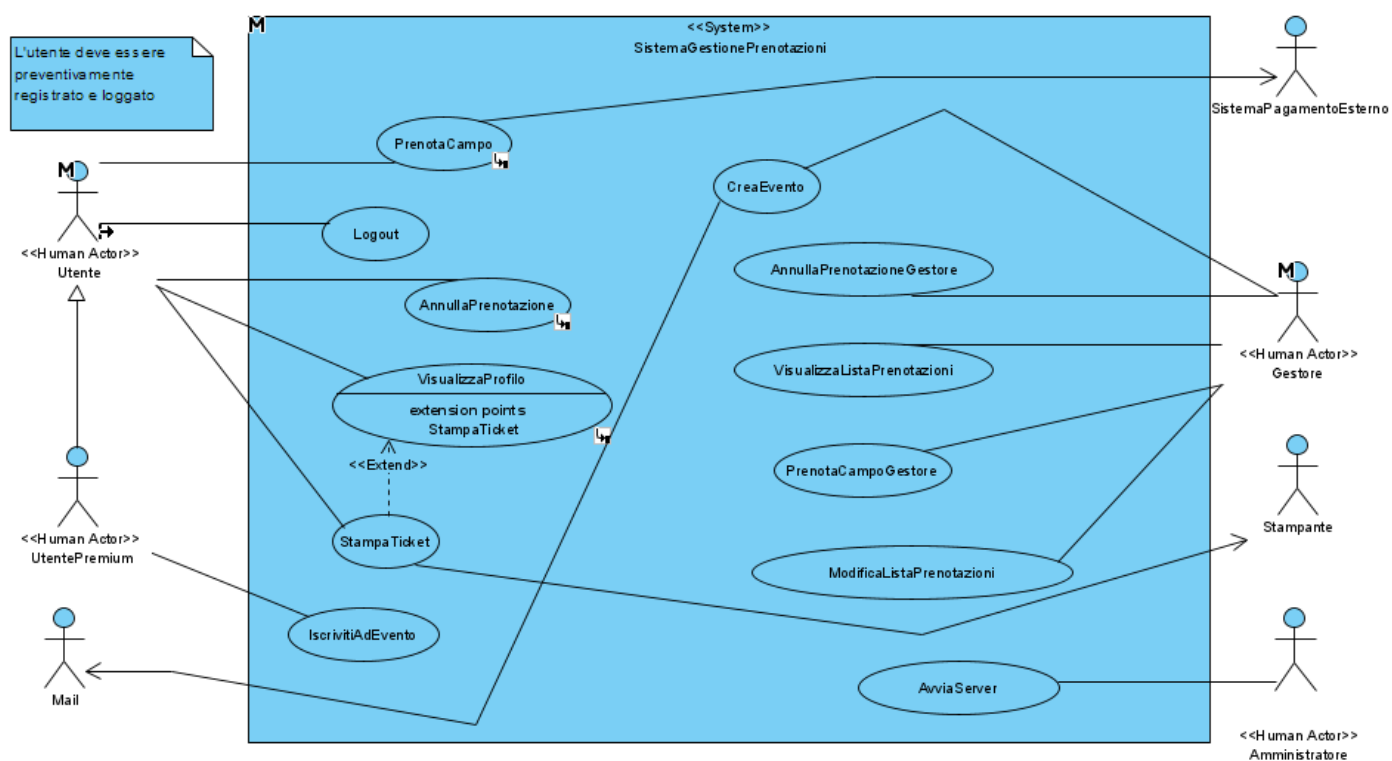
Nome caso d'uso	Login
Portata	Applicazione desktop
Livello	Sotto-funzione
Attore primario	Utente
Parti interessate	Utente (vuole accedere al sistema), Gestore (vuole che l'utente acceda al sistema per permettergli di usufruire delle funzionalità)
Pre-condizioni	L'utente deve essere registrato;
Garanzia di successo	L'utente accede al sistema; Il sistema memorizza l'utente che ha acceduto;
Scenario principale	<ol style="list-style-type: none"> 1. L'utente sceglie di accedere al sistema 2. L'utente inserisce username e password 3. Il sistema elabora la richiesta e memorizza l'id dell'utente che ha acceduto 4. Il sistema mostra un messaggio di avvenuto accesso correttamente
Estensioni	3.a: Username o password non corrette <ol style="list-style-type: none"> 1. Il sistema segnala l'errore 2. L'utente torna alla schermata di accesso

Nome caso d'uso	Logout
Portata	Applicazione desktop
Livello	Obiettivo Utente
Attore primario	Utente
Parti interessate	Utente (vuole fare il logout dal sistema), Gestore (vuole che l'utente possa fare il logout in modo da proteggere le proprie credenziali)
Pre-condizioni	L'utente deve essere registrato e deve aver acceduto al sistema;
Garanzia di successo	L'utente effettua il logout dal sistema; Il sistema è pronto per un nuovo accesso di un altro utente;
Scenario principale	<ol style="list-style-type: none"> 1. L'utente sceglie di fare il logout dal sistema 2. Il sistema elabora la richiesta 3. Il sistema mostra un messaggio di avvenuto logout correttamente
Estensioni	

Nome caso d'uso	Avvio server
Portata	Applicazione desktop
Livello	Obiettivo Utente
Attore primario	Amministratore
Parti interessate	Amministratore (vuole avviare il server), Utente (vuole che le prenotazioni vengano caricate per potervi accedere), Gestore (vuole che il sistema funzioni correttamente per poter guadagnare)
Pre-condizioni	Il server è funzionante e pronto per l'avvio
Garanzia di successo	Il server è avviato correttamente; Le prenotazioni per la giornata odierna sono caricate; Le prenotazioni della giornata precedente sono eliminate;
Scenario principale	<ol style="list-style-type: none"> 1. L'amministratore sceglie di avviare il server 2. Il sistema carica tutte le prenotazioni per la giornata odierna 3. Il sistema elimina tutte le prenotazioni della giornata precedente 4. Il server è avviato correttamente
Estensioni	

4.1.4 Diagramma dei casi d'uso

Il diagramma dei casi d'uso è necessario per esprimere un comportamento, desiderato o offerto. Individua chi o che cosa ha a che fare con il sistema (ossia l'attore) e che cosa l'attore può fare (ossia il caso d'uso). Tipicamente è il primo tipo di diagramma ad essere creato in un processo o ciclo di sviluppo nell'ambito dell'analisi dei requisiti.



Analisi dei costi

Gli Use Case Points permettono di misurare la dimensione di una applicazione e dedurre una stima sulla durata del progetto e i progressi del team. Il loro studio è essenziale per un team inesperto come il nostro, poichè ci ha permesso di stimare il numero di iterazioni necessarie per portare a termine il progetto.

1. Unadjusted Use Case Weight (UUCW)

Elenco casi d'uso con complessità stimata associata (complessità stimata in base al numero di transazioni, sia nello scenario normale che nei flussi alternativi). Ricordando che:

COMPLESSITA'	NUMERO TRANSAZIONI
semplice	$x < 4$
media	$4 \leq x \leq 7$
complessa	$x > 7$

CASO D'USO	COMPLESSITA'
PrenotaCampo(Utente)	Complessa
AnnullaPrenotazione(Utente)	Media
VisualizzaProfilo	Media
StampaTicket	Semplice
Registrazione	Semplice
Login	Semplice
Logout	Semplice
IscriviAdEvento	Media
PrenotaCampo(Gestore)	Media
AnnullaPrenotazione(Gestore)	Media
VisualizzaListaPrenotazioni	Semplice
ModificaListaPrenotazioni	Media
CreaEvento	Complessa
AvviaServer	Media

Tabella UUCW

COMPLESSITA'	PESO	NUMERO UC	PRODOTTO
Semplice	5	5	25
Media	10	7	70
Complessa	15	2	30

TOTALE=125

2. Unadjusted Actor Weight (UAW)

TIPO DI ATTORE	ESEMPIO	PESO
Semplice	Altro sistema tramite API	1
Medio	Altro sistema tramite protocollo oppure persona tramite interfaccia testuale	2
Complesso	Persona tramite interfaccia grafica	3

Elenco attori con tipo stimato associato:

ATTORE	TIPO
Utente	3
Utente Premium	3
Gestore	3
Amministratore	3
SistemaPagamentoEsterno	1
Stampante	1
SistemaMail	1

TIPO	PESO	NUMERO ATTORI	PRODOTTO
Semplice	1	3	3
Medio	2	0	0
Complesso	3	4	12

TOTALE=15

3. Unadjusted Use Case Points (UUCP)

$$\text{UUCP} = \text{UUCW} + \text{UAW} = 125 + 15 = 140$$

4. Aggiustamenti per la Complessità Tecnica (requisiti non funzionali)

FATTORI	PESO	VALUTAZIONI	IMPATTO
Sistema Distribuito	2	2	4
Obiettivi di Performance	2	3	6
Efficienza end-user	1	4	4
Elaborazione complessa	1	2	2
Codice riusabile	1	3	3
Facile da installare	0.5	1	0.5
Facile da usare	0.5	5	2.5
Portabile	2	1	2
Facile da modificare	1	3	3
Uso concorrente	1	3	3
Sicurezza	1	3	3
Accesso a terze parti	1	3	3
Necessità addestramento	1	1	1

TOTALE=37

Calcoliamo il Technical Complexity Factor, dopo aver ottenuto il TFactor dal Totale.

$$\text{TCF} = 0.6 + (0.01 \times \text{TFactor}) = 0.6 + (0.01 \times 37) = 0.97$$

5. Aggiustamento per la complessità dell'ambiente (ambiente di sviluppo)

Ci sono sia fattori migliorativi > 0 (che dovrebbero decrementare il conteggio degli use-case points) che fattori peggiorativi (che riducono il beneficio) < 0 .

FATTORE	PESO	VALUTAZIONE	IMPATTO
Familiare con il processo di sviluppo	1.5	1	1.5
Esperienza sull'applicazione	0.5	1	0.5
Esperienza sull'Object Orientation	1	3	3
Capacità dell'analista	0.5	2	1
Motivazione	1	4	4
Requisiti stabili	2	3	6
Staff part-time	-1	0	0
Linguaggio di programmazione difficile	-1	2	-2

TOTALE=14

Procediamo ora al calcolo dell'E-Factor secondo la seguente formula:

$$EF = 1.4 + (-0.03 \times EFactor) = 1.4 + (-0.03 \times 14) = 0.98$$

6. Calcolo Finale degli Use Case Points (UCP)

$$UCP = UUCP \times TCF \times EF = 140 \times 0.97 \times 0.98 \approx 133$$

7.Ore di lavoro totali

$$\text{UCP} \times 30 = 3990$$

Nota: Inizialmente, Karner propose un rapporto di 20 ore per UCP. Successivamente, Kirsten e Ribu raffinarono proponendo un valore tra 15 e 30 ore per UCP. Altri approcci tengono conto anche dei fattori ambientali per proporre rapporto variabile fra 20 e 28.

Poichè il nostro team è alla prima esperienza con UP si è deciso per il fattore moltiplicativo massimo, quindi 30.

8.Numero iterazioni

- 1 lavoratore = 30 ore a settimana

- 3 lavoratori = 90 ore a settimana

- iterazione di 4 settimane = 360 ore a iterazione

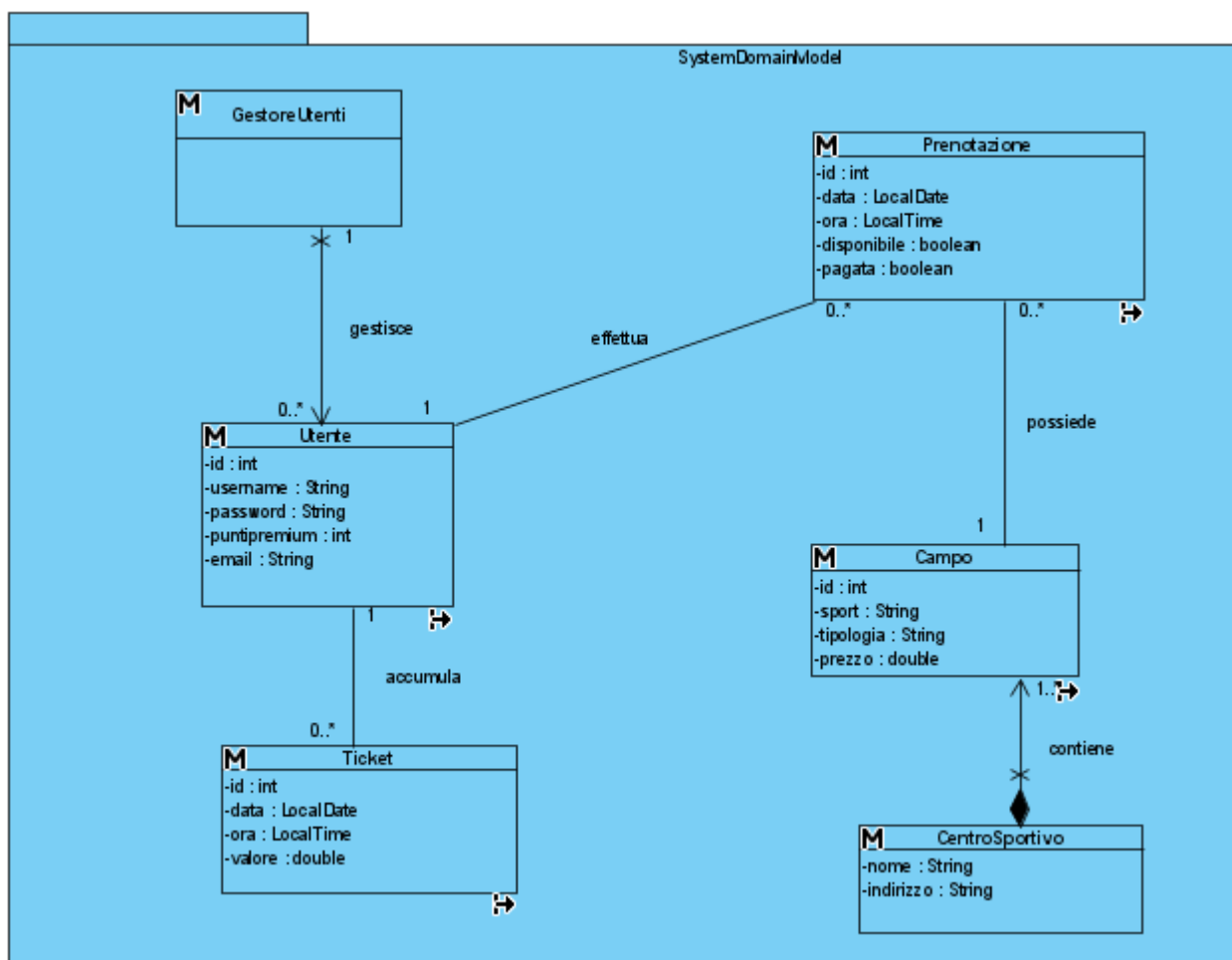
- ore di lavoro totale / ore a iterazione del gruppo di lavoro = $3990 / 360 = 11$

Quindi si stima che occorrano 44 settimane = 11 iterazioni per sviluppare questo progetto nella sua interezza.

4.2 System Domain Model

Il system domain model descrive le varie entità che fanno parte o hanno rilevanza nel sistema e le loro relazioni. Esso è stato utilizzato per la descrizione del nostro progetto al fine di mettere a fuoco i concetti fondamentali del sistema e definirne un vocabolario specifico.

Un importante merito dei domain model è infatti quello di fornire a tutti coloro che devono lavorare su un sistema una base comune di concetti su cui ragionare e una terminologia condivisa rigorosa e precisa.



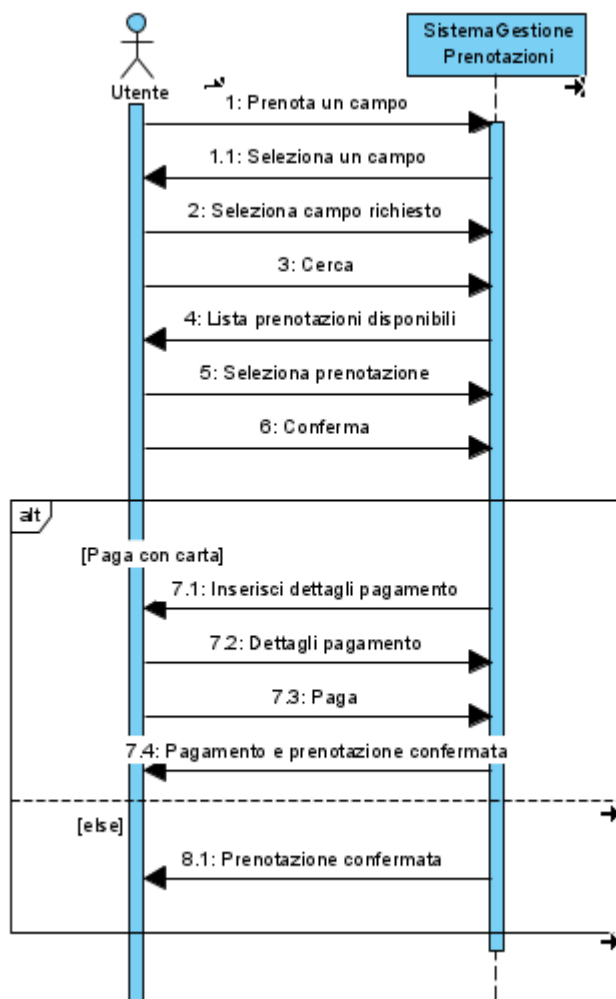
4.3 System Sequence Diagram

Un SSD è una vista del sistema che mostra, per un particolare scenario di caso d'uso, la sequenza di eventi generati dall'interazione dell'attore con il sistema.

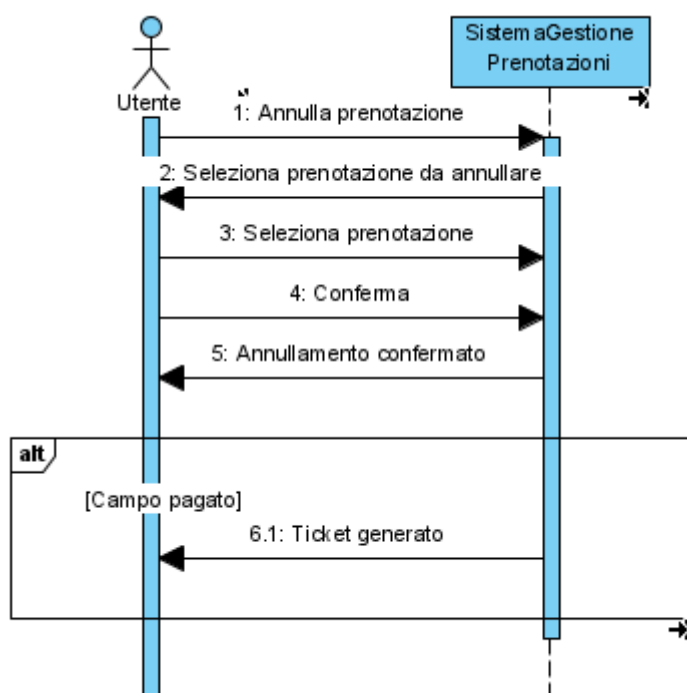
Per questo motivo la notazione utilizzata è quella tipica dei Sequence Diagrams.

Essi vengono mostrati per identificare le operazioni svolte dal sistema e specificarne le interazioni richieste.

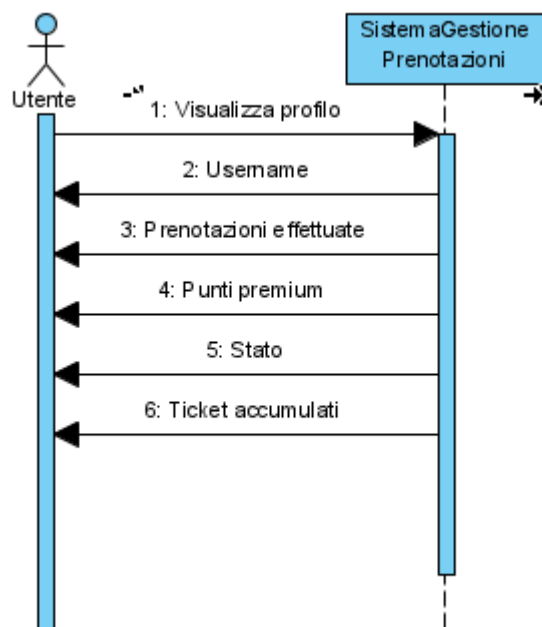
4.3.1 Prenota campo



4.3.2 Annulla prenotazione



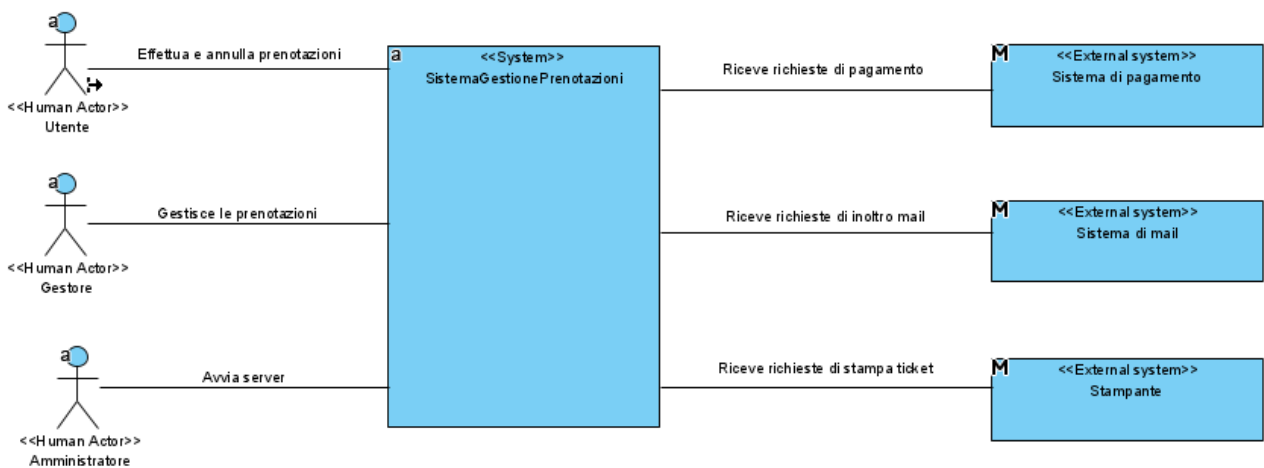
4.3.3 Visualizza profilo



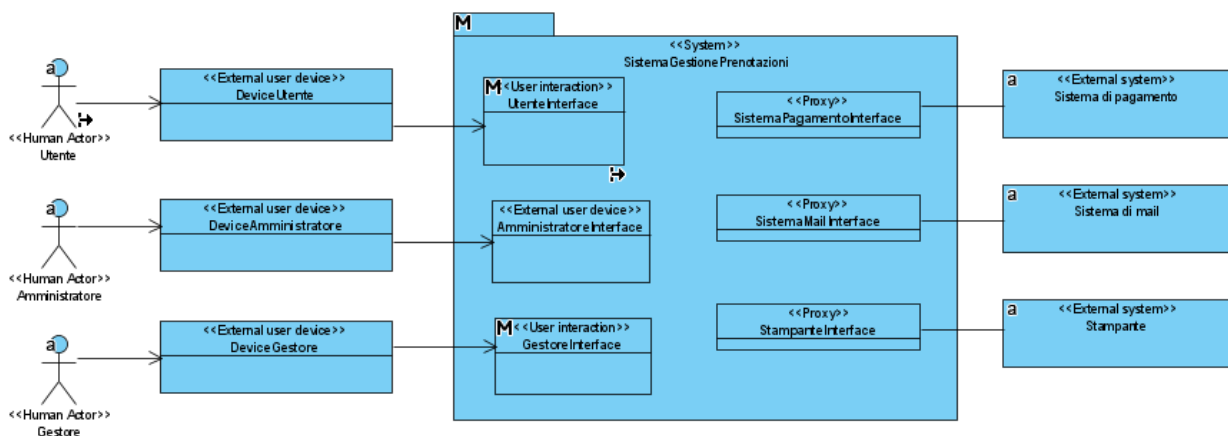
4.4 Context diagram

Il context diagram viene utilizzato per stabilire il contesto e i confini del sistema da modellare: esso chiarisce quali cose sono all'interno e quali all'esterno del sistema da modellare, definendone la relazione con il sistema. Nel nostro progetto sono stati utilizzati due diversi context diagram per specificare correttamente queste informazioni: si è utilizzato un primo diagramma per enfatizzare il tipo di relazione che ha il nostro sistema con le entità esterne; un secondo diagramma (detto anche context diagram con boundary) per introdurre le classi che hanno il compito di interfacciare l'utente al sistema e quelle che hanno il compito di gestire l'interazione con le entità esterne.

4.4.1 Context diagram



4.4.2 Context diagram con boundary



Capitolo 5: Documentazione di progetto

Questa fase ha come input la documentazione di analisi e ha il compito di aggiungere dettagli di progettazione e di sviluppo delle diverse soluzioni implementative.

5.1 Architettura logica

Lo stile architetturale prescelto per l'applicazione è quello Client-Server.

Il paradigma client-server è un modello di iterazione tra processi software, ove processi interagenti si suddividono tra client, che richiedono servizi, e server, che offrono servizi.

Il processo client è tipicamente dedicato ad interagire con l'utente finale; esso svolge un ruolo attivo, in quanto genera autonomamente richieste di servizi. Invece, il processo server è reattivo: esso svolge una computazione solo a seguito di una richiesta da parte di un qualunque client.

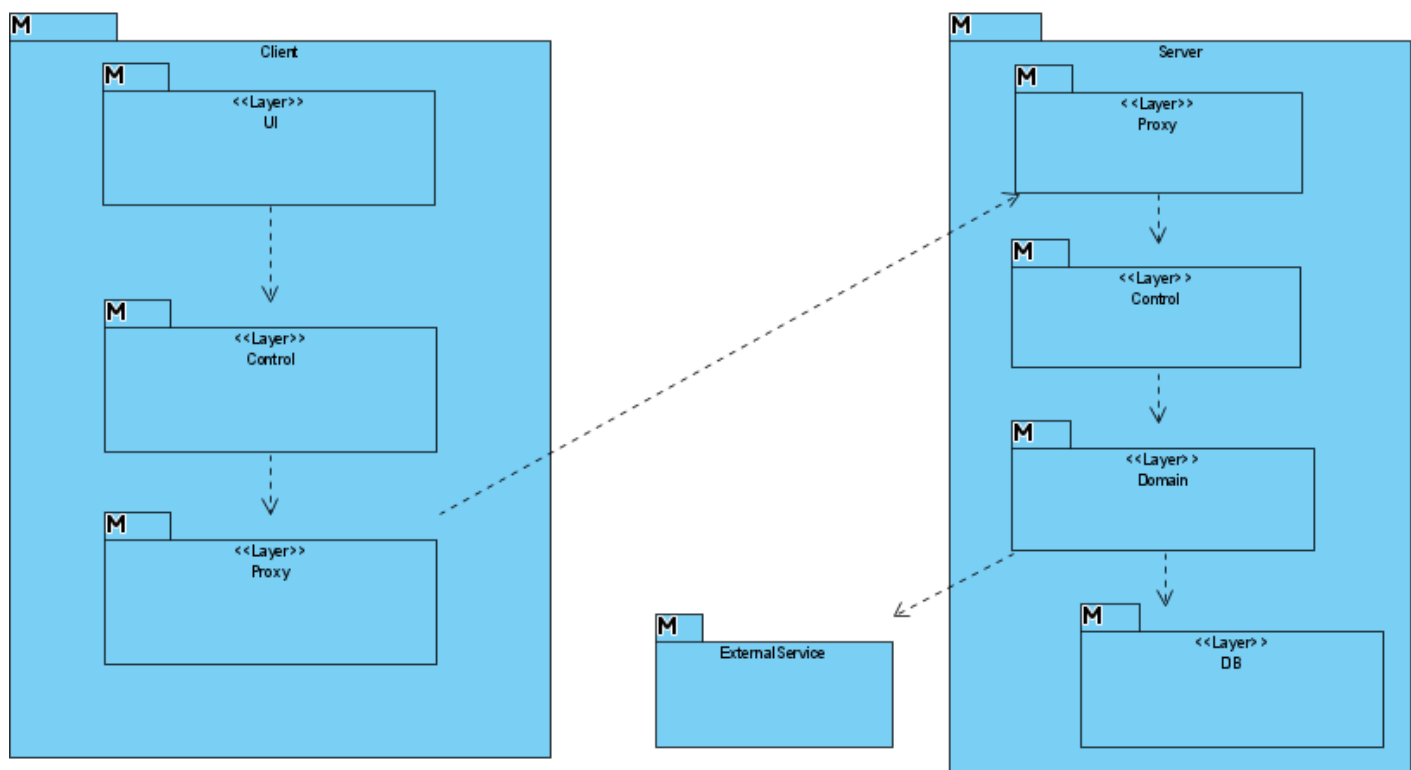
Normalmente, un processo client può richiedere in sequenza alcuni servizi a vari processi client.

5.2 Package Diagram

Nello sviluppo del progetto si è deciso di mantenere una gestione strict dei livelli.

Dal lato client il layer UI si occuperà della visualizzazione e dell'acquisizione degli input utente. Il layer control farà da gestore andando a coordinare le operazioni, interfacciandosi con un servizio di comunicazione remota che interagirà con il server.

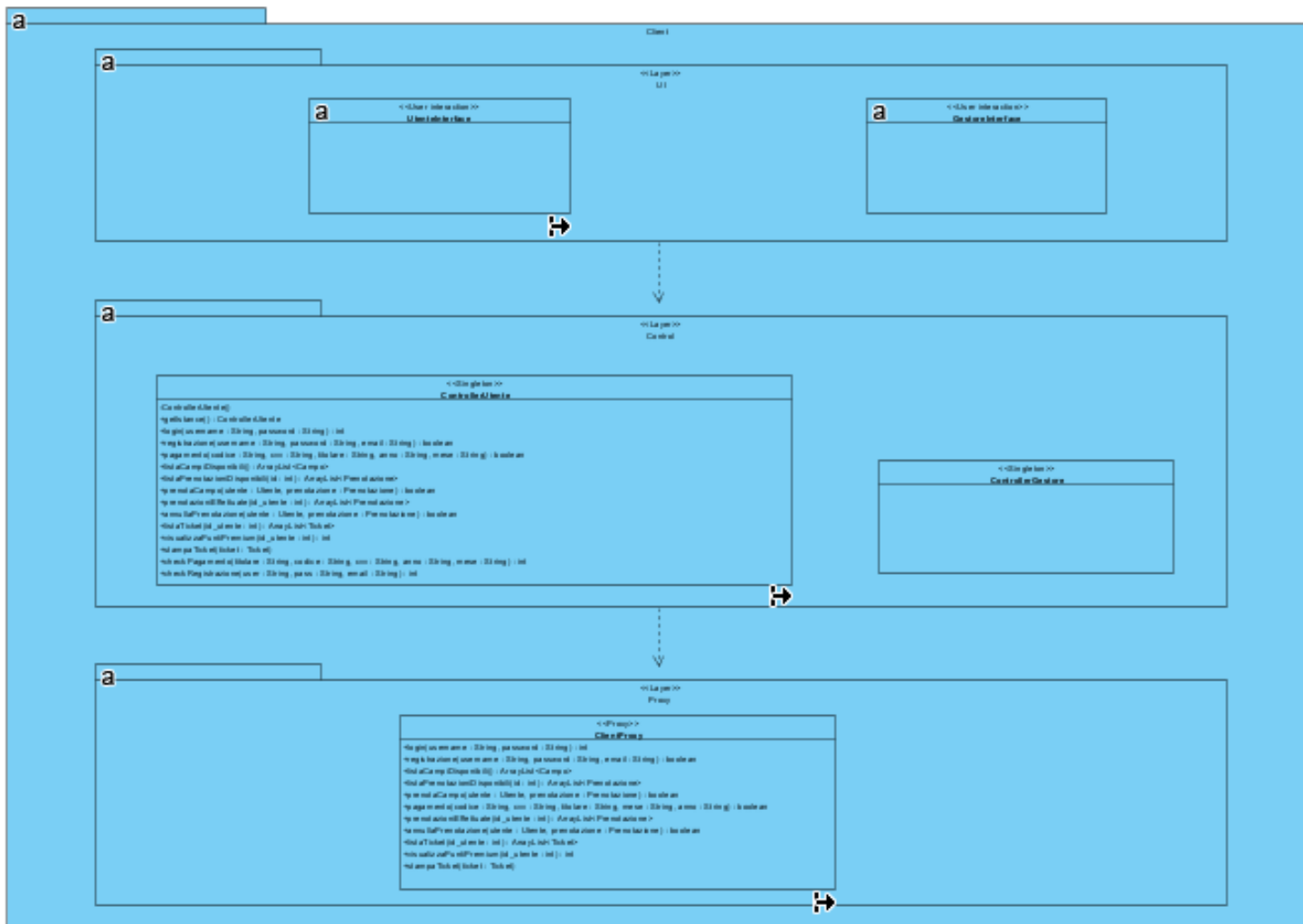
Dall'altro lato, invece, il server sarà in attesa di richieste, che verranno coordinate sempre a livello control. Le entità di dominio avranno poi compito di gestire la propria persistenza e le interazioni con i servizi esterni.



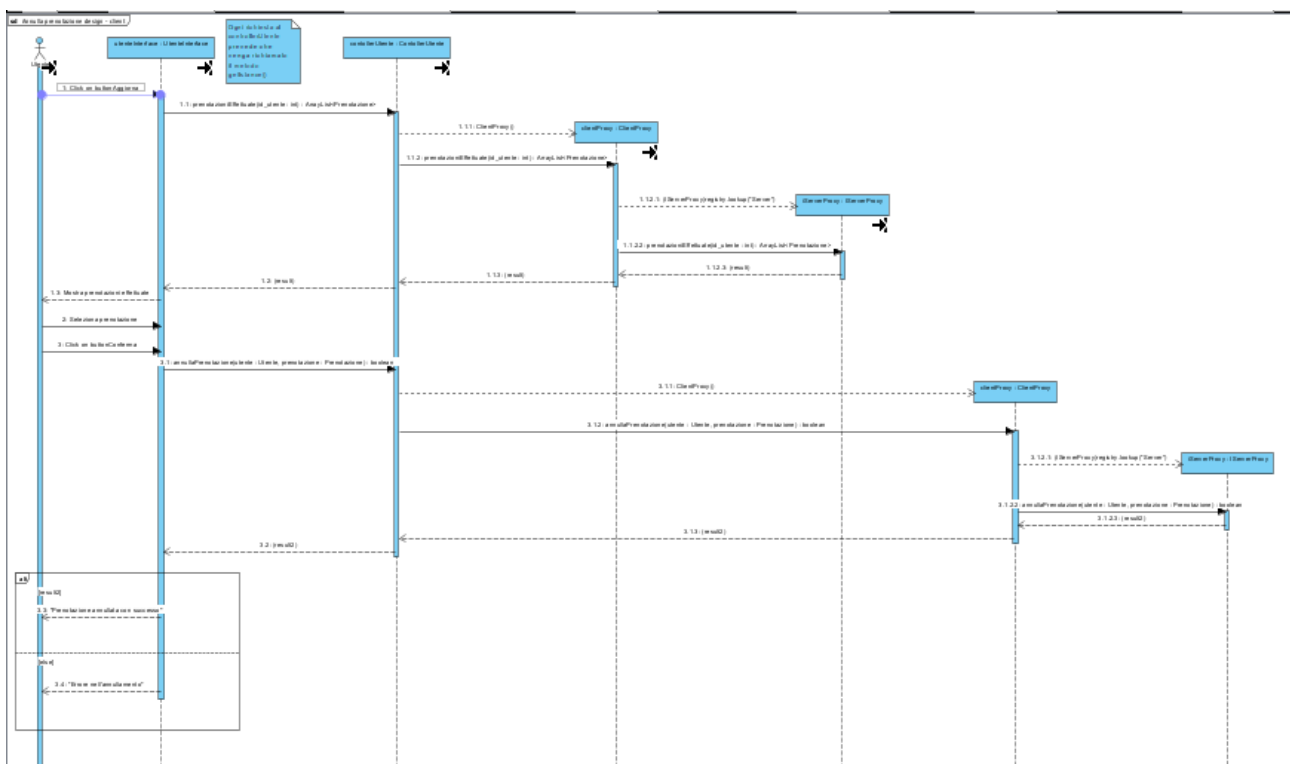
5.3 Class diagram

Per organizzare meglio la documentazione e renderla meno dispersiva, sono stati prodotti 3 class diagram di progettazione, uno per ogni macropackage (client server e external service).

5.3.1 Client class diagram

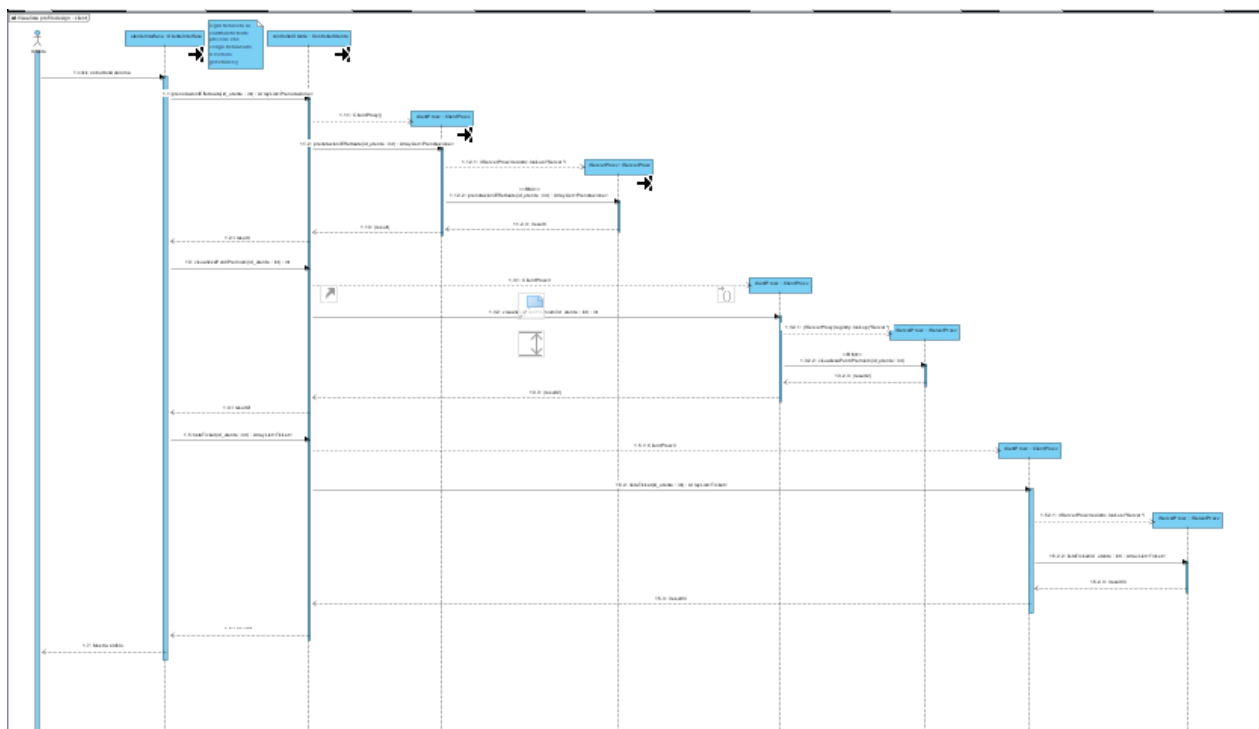


5.4.3 Annulla prenotazione client

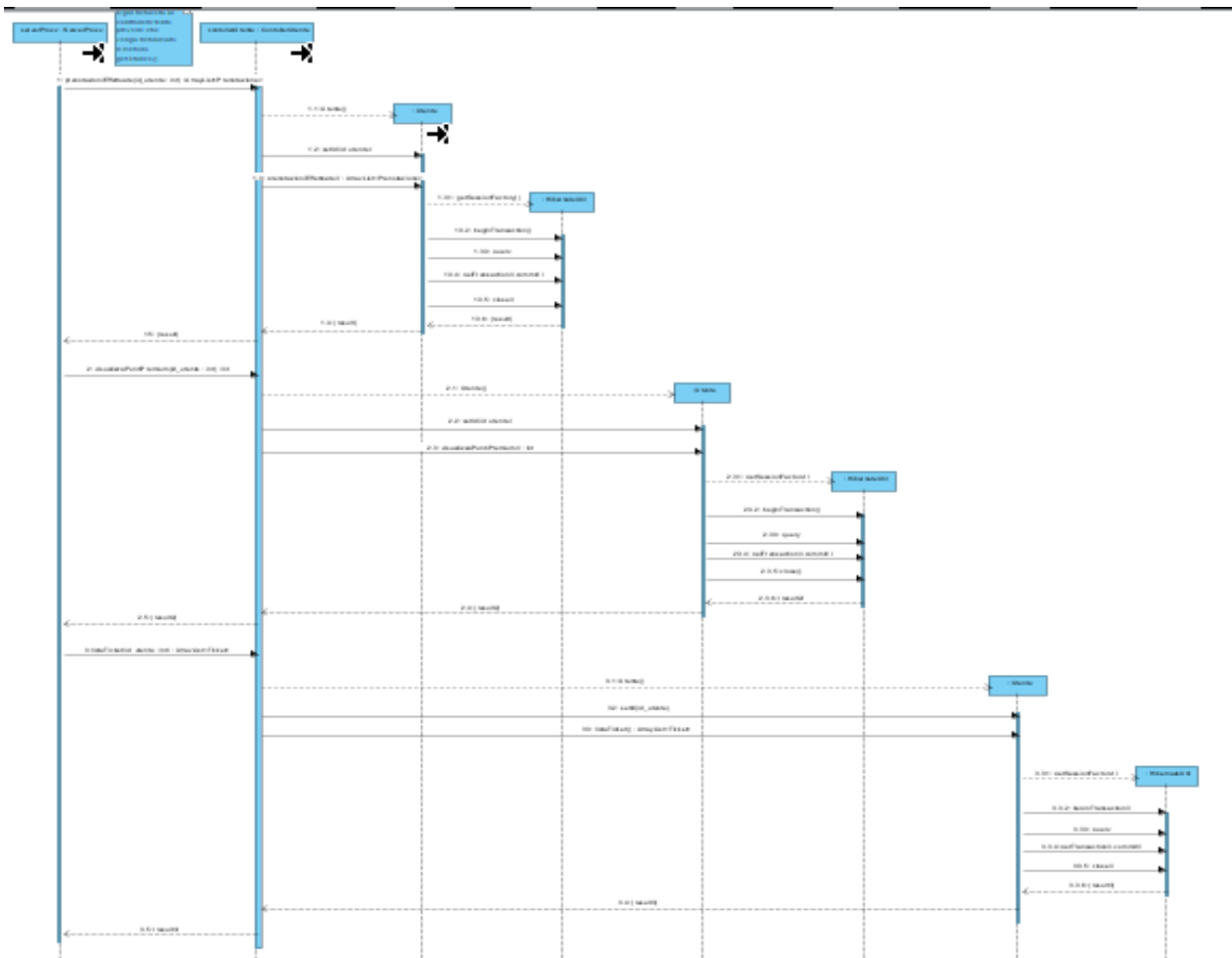


[illegible]

5.4.5 Visualizza profilo client



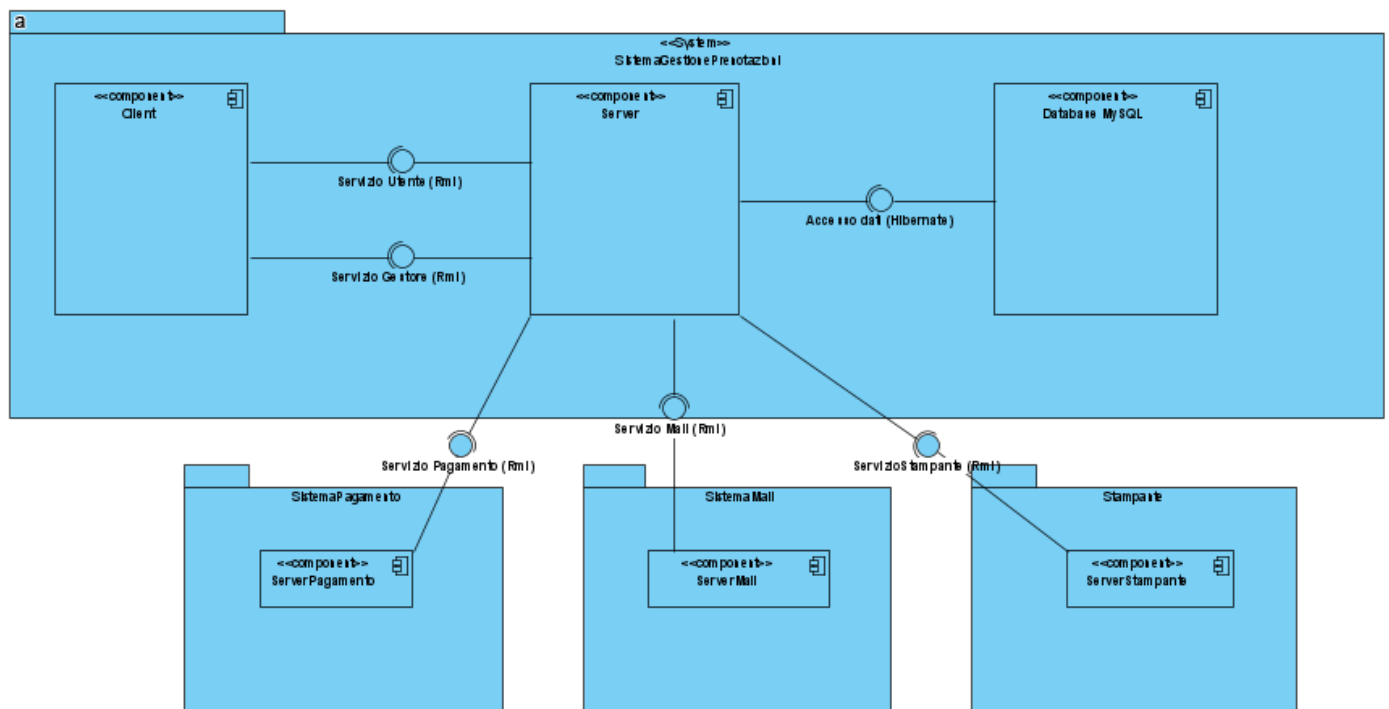
5.4.6 Visualizza profilo server



5.5 Component diagram

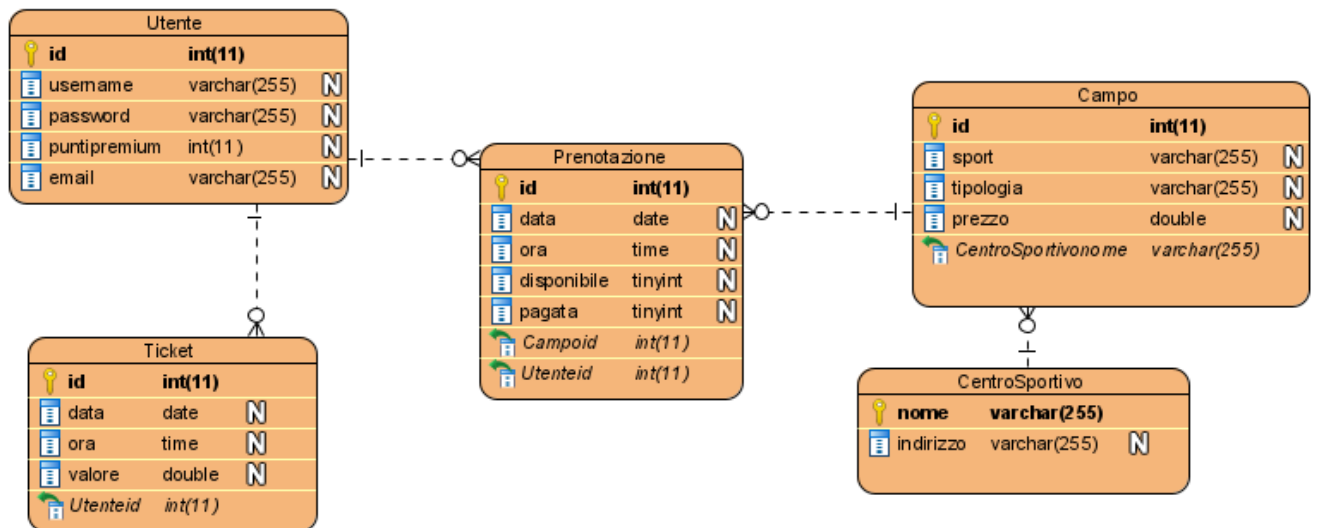
Il component diagram è un diagramma che ha lo scopo di rappresentare la struttura interna del sistema software modellato in termini dei suoi componenti principali e delle relazioni fra di essi. Per componente si intende un'unità software dotata di una precisa identità, nonché responsabilità e interfacce ben definite.

Esso è stato utilizzato nel nostro progetto per mettere in risalto le interfacce attraverso le quali comunicano i vari componenti sia con altri componenti presenti all'interno del sistema che con i componenti esterni al sistema.



5.6 Entity Relationship diagram

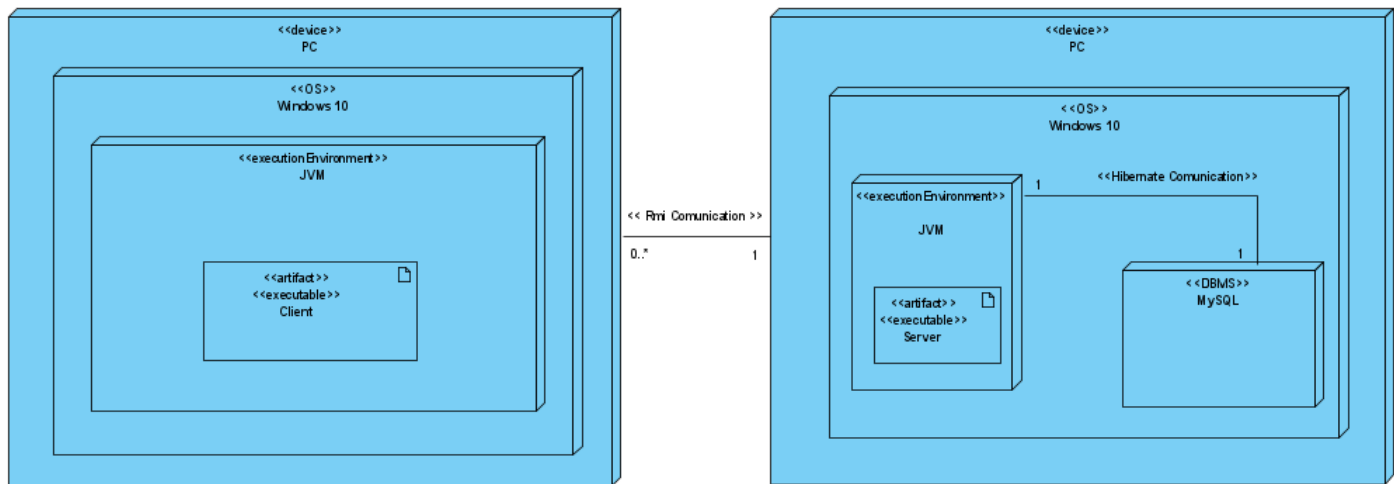
L'entity relationship diagram (ERD) mostra le relazioni dei set di entità archiviati in un database. Un'entità in questo contesto è un oggetto, un componente di dati. Un set di entità è una raccolta di entità simili. Queste entità possono avere attributi che ne definiscono le proprietà. Questo diagramma è stato utilizzato per abbozzare il progetto di un database, descriverne le entità in gioco e inoltre è il punto di partenza per produrre il codice SQL sfruttando tool automatici.



5.7 Deployment diagram

Il deployment diagram è un tipo di diagramma UML che mostra l'architettura di esecuzione di un sistema, inclusi nodi come ambienti di esecuzione hardware o software e il middleware che li collega.

Questo diagramma è stato utilizzato per visualizzare al meglio come le componenti software siano distribuite rispetto alle risorse hardware disponibili sul sistema e per capire come il sistema verrà distribuito fisicamente sull'hardware.



Capitolo 6: Implementazione

Completata la fase di progettazione è stato generato il codice sfruttando il tool preposto di Visual Paradigm. Successivamente a partire dal diagramma ER è stato generato il codice SQL per la creazione del database.

6.1 Strumenti e Framework a supporto utilizzati

Descriviamo ora i framework e gli strumenti utilizzati in fase di implementazione del progetto.

6.1.1 Visual paradigm

Visual Paradigm è uno strumento CASE UML che fornisce supporto per la modellazione e funzionalità di generazione automatica di codice per vari linguaggi di programmazione. E' stato utilizzato per generare automaticamente sia il codice JAVA delle varie classi a partire dal class diagram di progettazione, sia il codice SQL a partire dall'entity relationship diagram.

6.1.2 Eclipse

Eclipse è un ambiente di sviluppo integrato multi-linguaggio e multiplatforma, che può essere utilizzato per la produzione di software di vario genere.

Nel nostro progetto è stato utilizzato poiché è uno degli ambienti di sviluppo più completi ed utilizzati del mondo Java, e inoltre possiede un marketplace che permette l'aggiunta di un'infinità di espansioni per estendere le capacità del sistema.

6.1.3 Apache Maven

Apache Maven è uno strumento di gestione di progetti software basati su Java e build automation.

Maven effettua automaticamente il download di librerie Java e plug-in Maven dai vari repository definiti scaricandoli in locale o in un repository centralizzato lato sviluppo.

Questo permette di recuperare in modo uniforme i vari file JAR e di poter spostare il progetto indipendentemente da un ambiente all'altro avendo la sicurezza di utilizzare sempre le stesse versioni delle librerie.

Maven è stato selezionato come tool di build automation per il sistema poichè esso permette di configurare due aspetti fondamentali: la gestione delle dipendenze e la build.

Maven usa un file XML che descrive le dipendenze fra il progetto e le varie versioni di librerie necessarie nonché le dipendenze fra di esse.

Di seguito è riportato il contenuto del file utilizzato nel nostro progetto.

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">

  <modelVersion>4.0.0</modelVersion>
  <groupId>it.java.progettoPSSS</groupId>
  <artifactId>ProgettoPSSS</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <name>ProgettoPSSS</name>

  <dependencies>

    <dependency>
      <groupId>org.hibernate</groupId>
      <artifactId>hibernate-core</artifactId>
      <version>5.4.18.Final</version>
    </dependency>

    <dependency>
      <groupId>mysql</groupId>
      <artifactId>mysql-connector-java</artifactId>
      <version>8.0.21</version>
    </dependency>

  </dependencies>

</project>
```

6.1.4 Java RMI

Java RMI, o Remote Method Invocation, ha come scopo quello di rendere invisibile al programmatore la maggioranza dei dettagli della comunicazione su rete. La comunicazione avviene tramite degli stub, che comunicano attraverso delle interfacce predisposte dal fornitore di servizi. Il fornitore si registra sul cosiddetto registry, specificando il porto su cui vuole comunicare, e chi voglia accedere ai servizi forniti fa il lookup sul registry e invia richieste secondo l'interfaccia. Nel nostro progetto l'API Java RMI è stata utilizzata per la comunicazione sia tra client e server sia tra client e servizi esterni.

6.1.5 Window builder

Windowbuilder è un tool di creazione per interfacce grafiche in Java. E' composto sia da SWT Designer che Swing Designer e adotta una filosofia WYSIWYG che rende molto intuitiva il design. Ha un approccio event-driven che permette di specificare le risposte ad un'azione, attraverso una serie di listeners. Esso è stato utilizzato per le interfacce grafiche della nostra applicazione.

6.1.6 Hibernate

Hibernate è un middleware Open-Source che, attraverso il proprio framework, offre un servizio di ObjectRelational Mapping. Hibernate si pone a metà tra il livello della business logic dell'applicazione e l'RDBMS, offrendo tutti i servizi CRUD in una veste semplificata per lo sviluppatore. Nel nostro progetto è stato utilizzato per la gestione e la comunicazione con il database MySQL, ed è stato configurato utilizzando il seguente file .xml:

```
<?xml version='1.0' encoding='utf-8'?>
<!DOCTYPE hibernate-configuration PUBLIC
    "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
    "http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">

<hibernate-configuration>
  <session-factory>

    <property name="hibernate.connection.driver_class">com.mysql.jdbc.Driver</property>
    <property name="hibernate.connection.url">jdbc:mysql://localhost:3306/progettodb</property>
    <property name="hibernate.connection.username">root</property>
    <property name="hibernate.connection.password">admin</property>

    <property name="hibernate.show_sql">true</property>

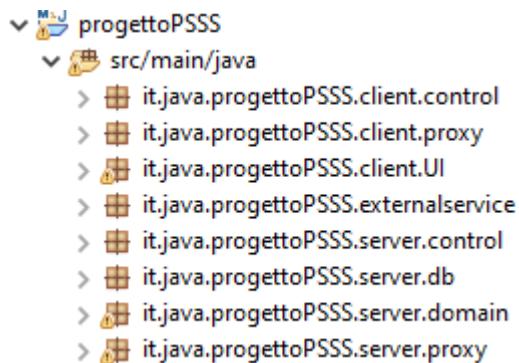
    <mapping class="it.java.progettoPSSS.server.domain.Utente" />
    <mapping class="it.java.progettoPSSS.server.domain.Campo" />
    <mapping class="it.java.progettoPSSS.server.domain.Ticket" />
    <mapping class="it.java.progettoPSSS.server.domain.Prenotazione" />
    <mapping class="it.java.progettoPSSS.server.domain.CentroSportivo" />

  </session-factory>
</hibernate-configuration>
```

6.2 Struttura del codice

Il codice è stato suddiviso nei seguenti package che rappresentano l'architettura logica del sistema:

- Client
- Server
- ExternalService



Nel package client sono contenuti:

- UI: insieme di classi necessarie per la gestione dell'interfacciamento con l'utente
- Control: insieme di classi necessarie per il controllo lato client
- Proxy: insieme di classi necessarie per la comunicazione del client con server e servizi esterni

Nel package server sono contenuti:

- Proxy: insieme di classi necessarie per la comunicazione del server con il client
- Control: insieme di classi necessarie per il controllo lato server
- Domain: insieme di classi che rappresentano le entità del dominio
- Db: insieme di classi per lo scambio di dati e la gestione del database

Nel package externalservice sono contenute le classi necessarie per la comunicazione tra servizi esterni e client.

6.3 Design pattern utilizzati

Per migliorare la scrittura del codice sono stati utilizzati i seguenti design pattern:

6.3.1 Singleton

Il pattern Singleton è un design pattern che limita la creazione di un'istanza di una classe a un'istanza "single". Tale pattern assicura che ci sia una sola istanza della classe, e fornisce un punto di accesso globale. Ciò è utile quando è necessario esattamente un oggetto per coordinare le azioni nel sistema. Esso è utile ad esempio quando si crea il contesto di un'applicazione, un driver per la connessione input/output con una console, un pool di thread e altre situazioni simili.

Nel progetto è stato utilizzato per rendere unica l'istanza delle classi controller, sia per il client sia per il server. Di seguito è riportata l'implementazione del pattern:

```
public class ControllerUtente {  
    private static ControllerUtente instance = null;  
  
    private ControllerUtente() {  
        super();  
    }  
  
    public static ControllerUtente getInstance() {  
        if (instance==null) {  
            instance = new ControllerUtente();  
        }  
        return instance;  
    }  
}
```

Si era pensato di utilizzare il singleton anche per quanto riguarda le classi proxy per la comunicazione tra client e server, ma è preferibile non fare abuso di questo pattern, per evitare di introdurre restrizioni non necessarie in situazioni in cui una sola istanza di una classe non è effettivamente richiesta.

6.3.2 Facade

Il pattern facade è un oggetto che permette, attraverso un'interfaccia più semplice, l'accesso a sottosistemi che espongono interfacce complesse e molto diverse tra loro, nonché a blocchi di codice complessi.

Nel nostro progetto è stato utilizzato in fase di avvio del server, poichè per far partire un server dovremmo essere a conoscenza di tutto il protocollo del server, richiamando tutte le funzioni necessarie, e questo implicherebbe una forte conoscenza del modello sottostante, ed è quello che si vuole evitare per mantenere ben separati i livelli.

In fase di avvio il server esegue le seguenti operazioni :

- Inizializzazione del server

```
public void startServer () {  
    try {  
        registry = LocateRegistry.createRegistry(1099);  
  
        ServerProxy server = new ServerProxy();  
        IServerProxy serverStub = (IServerProxy) UnicastRemoteObject.exportObject(server, 0);  
        Registry registry = LocateRegistry.getRegistry();  
        registry.rebind("Server", serverStub);  
        System.out.println("Server ready");  
    } catch (Exception e) {  
        e.printStackTrace();  
    }  
}
```

- Carica le prenotazioni per la giornata odierna

```
public void loadPrenotazioni () {

    LocalDate oggi = LocalDate.now();
    LocalTime ora = LocalTime.now();

    int da_creare = 23 - ora.getHour();

    CentroSportivo c = new CentroSportivo();

    ArrayList<Campo> campi = new ArrayList<Campo>();

    campi = c.listaCampiDisponibili();

    for (Campo campo : campi) {

        for (int i = 0; i < da_creare ; i++)
        {
            Prenotazione p = new Prenotazione();
            LocalTime t = LocalTime.of(ora.getHour()+i+1, 0);
            p.loadPrenotazione(oggi,t,campo);
        }
    }
}
```

- Elimina le prenotazioni delle giornate passate

```
public void deletePrenotazioni () {
    LocalDate oggi = LocalDate.now();
    LocalTime ora = LocalTime.now();

    CentroSportivo c = new CentroSportivo();

    ArrayList<Campo> campi = new ArrayList<Campo>();

    campi = c.listaCampiDisponibili();

    System.out.println("Campi trovati : " + campi.size());

    for (Campo campo : campi) {

        ArrayList<Prenotazione> pren = campo.listaPrenotazioniDisponibili();

        System.out.println("Prenotazioni trovati : " + pren.size());

        for (Prenotazione prenotazione : pren) {

            if (prenotazione.getData().getDayOfYear() < oggi.getDayOfYear() || (prenotazione.getData().getDayOfYear() == oggi.getDayOfYear()
                && prenotazione.getOra().getHour() <= ora.getHour())) {
                prenotazione.DeletePrenotazione(prenotazione.getId());
            }
        }
    }
}

}
```

Per rendere questo trasparente posso creare un facade.

```
public class ServerProxyFacade {  
    private final ServerProxy serverProxy;  
  
    public ServerProxyFacade (ServerProxy serverProxy) {  
        this.serverProxy = serverProxy;  
    }  
  
    public void startServer() {  
        serverProxy.startServer();  
        serverProxy.loadPrenotazioni();  
        serverProxy.deletePrenotazioni();  
    }  
}
```

Prendiamo tutte le chiamate di protocollo e le mettiamo all'interno di un metodo startServer.

All'esterno il client o qualsiasi utilizzatore non deve essere più a conoscenza del protocollo ma deve solo chiamare il metodo start.

Se in futuro cambiasse il server (e non avessi più queste funzioni ma altre come initializedb o initializeconnection ecc) all'esterno il client continuerà a chiamare startServer, in modo trasparente. C'è un unico punto di accesso del tutto indipendente dal modello sottostante.

Capitolo 7: Testing

Per quanto riguarda la fase di testing si è deciso di utilizzare il framework integrato in Eclipse JUnit.

JUnit è un framework open-source che consente di scrivere unit test, supportando chi sviluppa software Java nell'attività di testing. Esso consente di scrivere ed eseguire automaticamente Test Case e Test Suite e permette di elaborare test di oggetti e classi Java.

Per il testing del progetto è stata utilizzata la versione Junit4 di Eclipse.

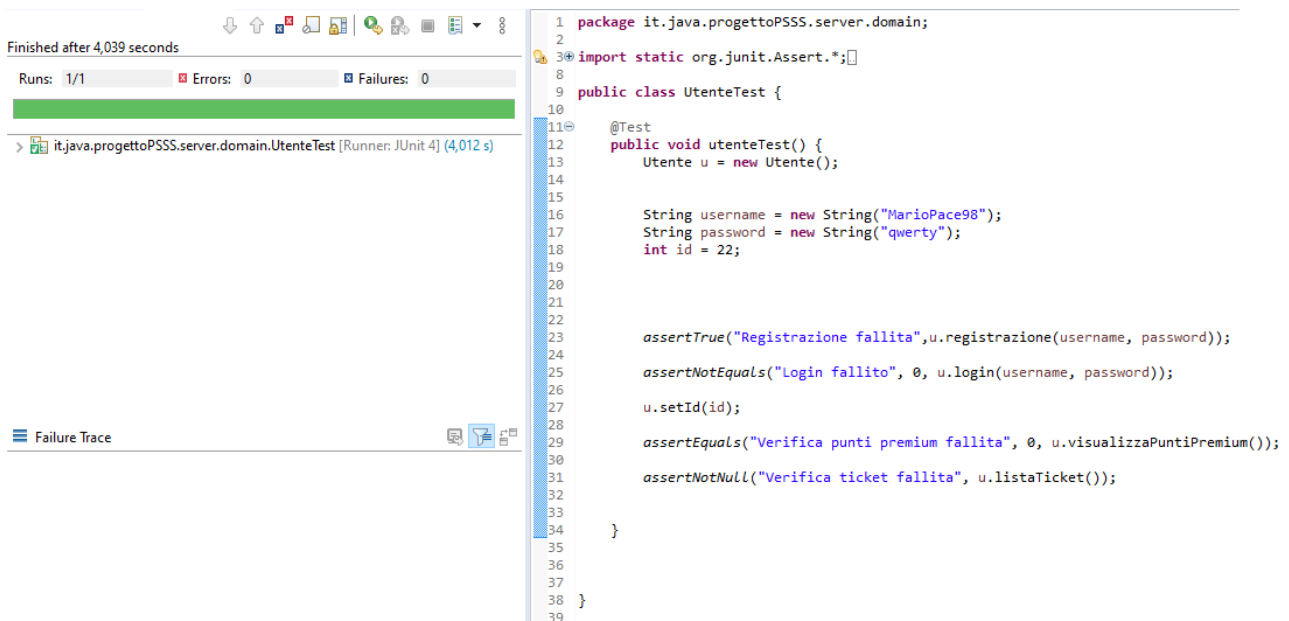
7.1 Testing Classe Utente

Nella prima iterazione è stato completato il testing per quanto riguarda le funzioni della classe Utente.

Sono state testate 4 funzioni in particolare:

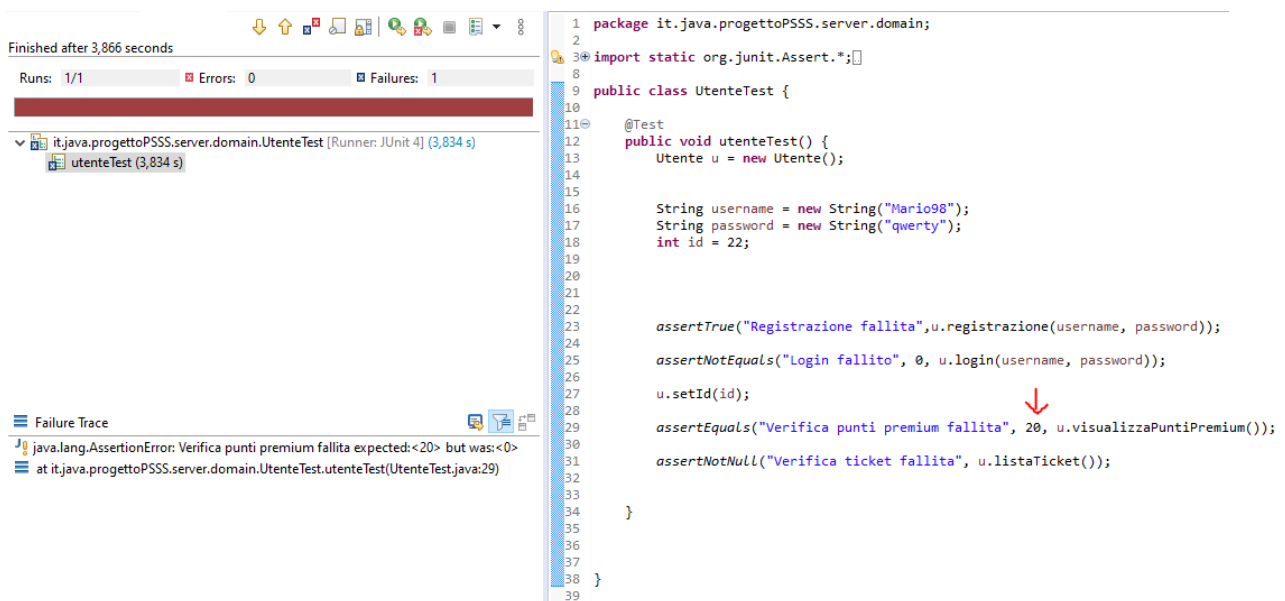
- Registrazione: si è verificato che la funzione registrazione restituisca true dando in input username e password univoci.
- Login: si è verificato che la funzione login restituisca un id diverso da 0, che implica che l'accesso è stato completato con successo.
- visualizzaPuntiPremium: si è verificato che i punti premium di un utente appena registrato siano 0.
- listaTicket: si è verificato che la funzione restituisca un oggetto non nullo e quindi effettivamente una lista (anche vuota come in questo caso).

7.1.1 Test successo



Il test ha successo con gli input dati, come possiamo notare dalla barra verde sulla sinistra.

7.1.2 Test fallito



In questo caso il test fallisce perché il confronto nell'`assertEquals` è fatto con 20 (e non più con 0) e ovviamente un utente appena registrato non può avere già 20 punti premium.

Sviluppi futuri

La prima iterazione della nostra desktop application presenta ancora una serie di scenari da raffinare e perfezionare in sviluppi futuri.

In primo luogo ci si propone di inserire maggiori controlli durante la registrazione dell'utente al fine di evitare malintenzionati.

Inoltre si prospetta di inserire la possibilità di effettuare pagamento anche in contanti, prevedendo quindi un ulteriore interfacciamento con un sistema di pagamento.

Qualora il centro sportivo dovesse decidere di aggiungere più postazioni sulle quali effettuare le prenotazioni, si dovrà prevedere una gestione della concorrenza nel caso di due o più prenotazioni simultanee.