

DOCUMENTAZIONE

AMDD: Iterazione 0

1.1 Introduzione al progetto

Il sistema si occuperà della gestione dei centri sportivi della società NewSport S.R.L., situata sul suolo di Milano e può contare su un centro in cui si trovano campi per praticare due sport: calcio a 5 e padel.

Il sistema viene utilizzato da tre tipologie di utente:

- Organizzatore, una figura che si occupa di creare eventi per un determinato impianto (specificando data e ora), le cui partite verranno generate automaticamente dal sistema in base alla disponibilità dei campi;
- Amministratore, viene selezionato dall'organizzatore per gestire lo stato dei campi, ad esempio se sono disponibili o necessitano di una manutenzione e quindi sono temporaneamente inaccessibili.
- I giocatori (i clienti), possono effettuare una scelta tra:
 - unirsi a un evento esistente, a patto che il numero di persone che vuole iscriversi insieme a lui sia minore o uguale al numero di posti disponibili;
 - prenotare privatamente un campo, in base anche alla presenza o meno di eventi, senza però dare la possibilità a giocatori non autorizzati d'iscriversi.

Di seguito viene descritto in breve il ciclo di vita di un evento:

- I. Un organizzatore crea un evento (data, ora, luogo);
- II. ci si può iscrivere all'evento entro la data di scadenza (da definire). Nel caso in cui raggiunta la data di scadenza non si è a sufficienza l'evento verrà annullato (da rilasciare);
- III. se l'evento viene confermato si mandata una notifica a tutti gli iscritti.

L'iscrizione a un evento avviene nel seguente modo:

- I. Si effettua una ricerca secondo parametri a piacere (zona/sport/data) e viene mostrata la lista di eventi disponibili (da vedere);
- II. selezionando un evento vengono visualizzati i posti disponibili, e se il gruppo che si vuole iscrivere ha un numero inferiore rispetto al numero di posti rimanenti, è possibile iscriversi.

Per gestire le scadenze nel caso in cui non ci sia un numero sufficiente di giocatori viene spuntata una casella in fase di creazione dell'evento che rappresenta il consenso dei giocatori verso un numero flessibile di partecipanti all'evento. Nel caso questa casella non venga spuntata e non viene raggiunto il numero di giocatori alla scadenza, l'evento viene annullato.

L'organizzatore ha il compito di creare l'evento (specificando la scadenza per l'iscrizione) e inserirlo nella lista attesa. Successivamente, il sistema alloca i partecipanti in modo da occupare tutti i posti per uno o più campi e invia una notifica a ogni utente che viene aggiunto a una partita.

1.2 ToolChain

Tool	Utilizzo
Eclipse	IDE per sviluppo back-end
Spring	Framework java per API/backend
Android Studio (o vue)	Sviluppo front-end
Java	Linguaggio di programmazione front e back end
StarUML	Grafici UML
Postman	Testing funzionamento API
JUnit	Framework per analisi dinamica del codice
CodeMR	Tool per analisi statica del codice
Draw.io	Tool per creazione di grafici
GitHub	Piattaforma per versionamento e condivisione codice e documentazione
Google Drive	Condivisione documentazione in fase di stesura
Discord	Applicazione per comunicazione a distanza e riunioni
Notion	Software per gestione e coordinazione delle task

CodeTogether	Plugin per scrittura in simultanea del codice in Eclipse
--------------	--

1.3 Requisiti Funzionali

ID	Titolo	Priorità
UC1	Registrazione	Alta
UC2	Login utente e amministratore/organizzatore	Alta
UC3	Logout utente e amministratore/organizzatore	Alta
UC4	Gestione campi dell'amministratore	Alta
UC5	Gestione prenotazioni (inserimento prenotazione privata, cancellazione prenotazione privata, visualizzazione di tutte le partite private/eventi) dell'utente	Alta
UC6	Gestisci amministratori (da parte del organizzatore)	Media
UC7	Organizzatore gestisce eventi (creazione/ annullamento)	Alta
UC8	Iscrizione ad evento	Alta
UC9	Creazione algoritmica squadre di evento	Alta
UC10	Annullamento prenotazione privata (amministratore)	Media
UC11	Notifica conferma evento	Media
UC12	Visualizzazione dei campi (disponibilità)	Media
UC13	Visualizzazione prenotazioni effettuate utente	Media
UC14	Valutazione campi utente	Bassa
UC15	Sistema ranking campi	Bassa
UC16	Gestione profilo utente	Bassa

1.4 Requisiti non funzionali

Usabilità

Si vuole sviluppare un'applicazione che abbia un front-end pensato per un utilizzo immediato anche per utenti poco esperti. In particolare si vuole creare un app mobile intuitiva e semplice in modo da invogliare l'utente a usufruire dei servizi offerti dalla società.

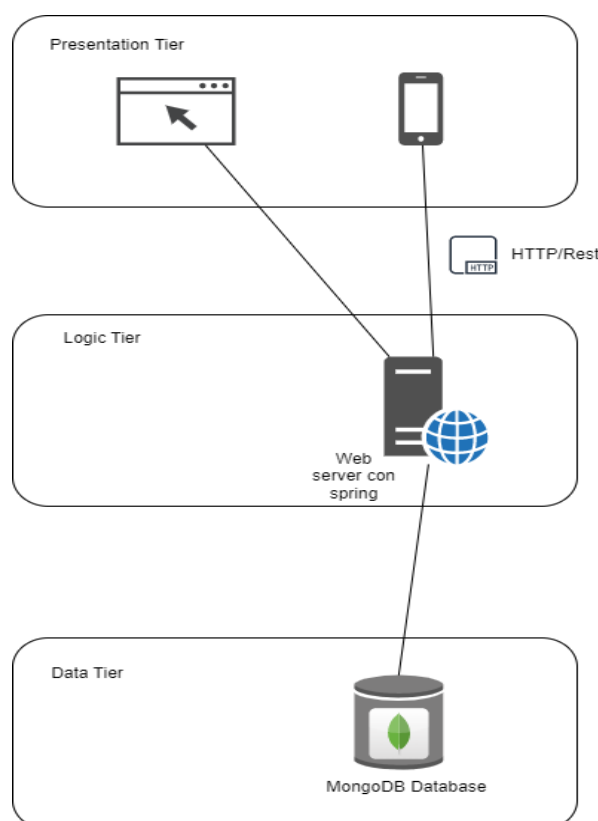
Manutenibilità

Nello sviluppo di questo software sarà importante scrivere codice semplice da leggere, interpretare e il più possibile scomposto in componenti. In questo modo l'applicazione sarà più flessibile a modifiche future e faciliterà l'intervento anche da parte di nuovi sviluppatori.

Efficienza

Il sistema offrirà la possibilità anche a chi non raggiunge il numero necessario per iniziare una partita, di giocare, attraendo in questo modo il maggior numero di clienti. Le partite con un numero massimo di giocatori saranno privilegiate in modo da ottimizzare al massimo la gestione dei campi.

1.5 Architettura del sistema



Si tratta di una architettura a tre livelli:

- Livello Presentazione (lato client): si tratta di una interfaccia per l'utente
- Livello Logico (lato server): contiene la logica del sistema con le relative API
- Livello Dati (lato server): database in cui vengono salvati i dati

o quindi in tre layer.

1.6 Design pattern MVP

Sviluppiamo il software seguendo il design Model View Presenter, dividendo:

- View: per visualizzare i dati e gestire l'interazione con l'utente
- Model: definisce le API per fornire i servizi offerti dall'applicazione e accedere al database
- Presenter: Gestisce la logica d'interazione con le API e permette la comunicazione tra Model e View.

Nel design MVP, la View e il Model sono scollegati, in questo modo possono essere sviluppati indipendentemente.

1.7 Use Case Stories

In questa sezione verranno presentate le use-case stories al fine di riuscire a creare uno Use-case diagram che possa soddisfare tutti i requisiti desiderati dal cliente.

Queste Use case stories sono divise in base agli attori che interagiscono con il sistema. Verranno brevemente descritte tutte le funzionalità che il cliente si aspetta di poter usufruire nell'utilizzo dell'applicazione.

Sono stati individuati tre attori che interagiranno con il sistema:

- Utente base: composto dai singoli giocatori che hanno un accesso limitato alle informazioni del sistema;
- Amministratore impianto: colui che presiede un impianto ed ha la responsabilità di gestirne la manutenzione e supportare i clienti durante lo svolgimento dell'attività;
- Organizzatore: una sorta di super-user che è in grado di organizzare eventi e rappresenta un gestore di tutti gli impianti distribuiti.

1.7.1 Use case stories: user base

Registrazione alla piattaforma

Come utente base, voglio potermi registrare nella piattaforma così da poter interagire con il sistema.

Login/logout

Come utente base, voglio poter fare login/logout dal mio account personale in cui risiedono le mie informazioni private.

(Gestione profilo utente) Visualizzazione e modifica informazioni profilo

Come utente base, voglio poter visualizzare le informazioni presenti all'interno del mio account personale, modificare le informazioni presenti sul mio account (nome profilo, foto profilo, età, sesso, ...). Inoltre voglio poter ricevere notifiche o email relative a una prenotazione avvenuta con successo a un particolare evento oppure a una partita singola.

Valutazione campo

Come utente base, voglio poter valutare il servizio ricevuto durante lo svolgimento della partita. Si tratta di una valutazione generale che deve tenere conto della qualità della

struttura, stato dei campi di gioco, cordialità degli amministratori, costo, e altri parametri simili.

Gestione prenotazioni / Iscrizione evento

Come utente, voglio essere in grado di vedere i campi disponibili e gli eventi disponibili in una determinata data.

Dunque, voglio poter essere in grado di prenotarmi (entro la scadenza) agli eventi o prenotare tutto il campo per una partita privata, ed eventualmente cancellarmi (entro una certa data di scadenza).

Visualizzazione disponibilità campi ed eventi

Come utente, voglio poter essere in grado di vedere quali sono i campi liberi prenotabili e gli eventi disponibili a cui posso partecipare, anche senza necessariamente effettuare il login.

1.7.2 Use case stories: amministratore impianto

Gestione profilo amministratore

Come amministratore, voglio essere in grado di modificare il mio profilo personale e visualizzare le informazioni presenti all'interno del mio profilo.

Gestione campi

Come amministratore, voglio essere in grado di rendere accessibili alla prenotazione i campi nell'impianto da me presieduto

Come amministratore, voglio poter togliere dalla disponibilità un campo, per esempio per la manutenzione, la pulizia o per eventi straordinari

Come amministratore, voglio poter inserire campi nel caso in cui vengano aggiunti nuovi spazi all'impianto da me presieduto.

Come amministratore, voglio poter cancellare i campi nel caso in cui questi non possano essere più utilizzati all'interno della struttura.

Gestione prenotazioni

Come amministratore voglio poter essere in grado di cancellare o aggiungere prenotazioni straordinarie da parte di utenti.

Login/logout

Come amministratore, voglio poter far login/logout dalla mia pagina personale.

1.7.3 use case stories: Organizzatore

Login/logout

Come organizzatore, voglio poter avere la possibilità di fare login/logout con delle credenziali particolari che mi permettano di gestire l'intero sistema come una sorta di super-user.

Registrazione amministratori

Come organizzatore, voglio poter registrare all'interno del sistema i vari amministratori che presiedono gli impianti, fornendo poi agli amministratori le credenziali per l'accesso.

Gestione evento

Come organizzatore, voglio poter essere in grado di creare, modificare e cancellare un evento, potendo scegliere la data dell'evento, la scadenza relativa alle prenotazioni e il numero di partecipanti.

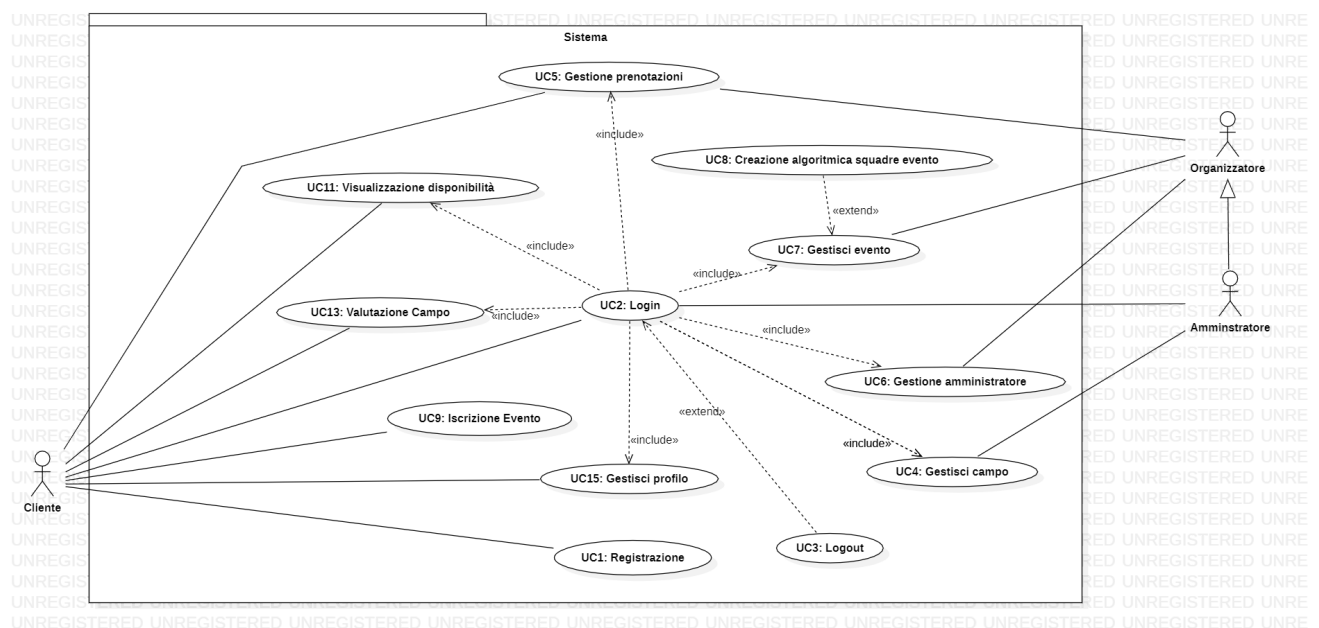
1.8 UML Use Case Diagram

Dalla specifica dei requisiti e dalle use case stories, è stato creato un diagramma dei casi d'uso che rappresenta l'interazione tra gli attori con il sistema.

Dalle use case stories sono stati individuati 3 attori principali

- Utente base
- Amministratore
- Owner

Sotto è riportata l'immagine del diagramma in questione.



AMDD: ITERAZIONE 1

In quest'iterazione verranno implementati i seguenti casi d'uso:

- UC1: Registrazione utente
- UC2: Login utente
- UC3: Logout utente
- UC6: Gestione campi
 - UC6.1: Inserimento campi
 - UC6.2: Visualizzazione campi
 - UC6.3: Cancellazione campi
 - UC6.4: Modifica campi
- UC7: Gestione eventi
 - UC7.1: Visualizza eventi
 - UC7.2 Inserisci evento
 - UC7.3: Cancella evento
 - UC7.4: Modifica evento

2.1 UC1: Registrazione utente

Descrizione:

Utente inserisce le informazioni personali in un form per iscriversi al servizio. I suoi dati vengono inseriti nel database. Viene eseguita una verifica sulla validità dei valori inseriti.

Attori coinvolti:

Utente, Sistema.

Trigger:

Invio del comando per la registrazione

Postcondizione:

I dati dell'utente sono salvati nel database e l'utente potrà effettuare il login.

Procedimento:

1. L'utente inizia il processo di registrazione
2. L'utente compila un form inserendo i suoi dati personali (Nome, Cognome, Email, Password)
3. Il sistema, se i valori inseriti nei campi del form sono validi, aggiunge l'utente al db.

2.2 UC2: Login utente

Descrizione:

L'utente inserisce i dati all'interno del form per effettuare il login; viene fatto un check da parte del sistema per la correttezza delle informazioni inserite, se corrispondono a quelle presenti nel database, il sistema permette l'accesso

Attori coinvolti:

Utente, Sistema

Trigger:

L'utente richiede di fare il login

Postcondizione:

L'utente viene indirizzato nella propria pagina personale

Procedimento:

1. L'utente inizia la procedura per fare il login

2. L'utente specifica il proprio username e la propria password
3. Il sistema controlla la correttezza delle informazioni inserite e ci sono due possibilità
 - a. Le credenziali sono corrette, il sistema consente l'accesso
 - b. le credenziali non coincidono con nessun record nel database: l'accesso viene negato

2.3 UC3: Logout utente

Descrizione:

L'utente effettua il logout dalla piattaforma

Attori coinvolti:

Utente, Sistema

Trigger:

L'utente manda il comando per effettuare il logout

Postcondizione:

L'utente non ha più accesso alle informazioni e viene reindirizzato alla schermata iniziale per effettuare nuovamente il login.

Procedimento:

1. L'utente manda il comando per il logout
2. L'utente viene scollegato dal sistema e reindirizzato alla pagina iniziale.

2.4 UC6: Gestione impianto

2.4.1 UC6.1: Inserimento campi

Descrizione:

L'amministratore aggiunge un campo all'elenco relativo all'impianto indicando nome, tipologia di sport.

Attori coinvolti:

Amministratore, Sistema

Trigger:

Amministratore invia un comando per la creazione di un campo

Postcondizione:

Tutti gli Utenti e gli Organizzatori che accedono alla piattaforma potranno vedere un nuovo campo aggiunto nell'elenco relativo all'impianto, ed eventualmente prenotarsi

Procedimento:

1. L' Amministratore effettua il Login e richiede la creazione di un nuovo campo
2. Il Sistema mostra una schermata con delle informazioni da completare come il nome del campo, il tipo di sport, ecc...
3. L' Amministratore compila i campi
4. L' Amministratore conferma l'azione e il campo viene aggiunto all'elenco dei campi esistenti nell'impianto in questione.

2.4.2 UC6.2: Visualizzazione campi

Descrizione:

Ogni utente, sia che abbia effettuato il Login o meno può vedere la lista dei campi disponibili e di quelli occupati al momento sulla piattaforma, senza però procedere con la prenotazione.

Attori coinvolti:

Utente, Sistema.

Trigger:

Mostrato immediatamente all'accesso nel sistema

Postcondizione:

Procedimento:

1. L'utente avvia l'applicazione
2. Il sistema mostra l'elenco dei campi disponibili al momento.

2.4.3 UC6.3: Cancellazione campi

Descrizione:

L'amministratore può eliminare un campo da un'elenco e non renderlo più disponibile, sia in modo definitivo che temporaneo, ad esempio a causa di manutenzioni periodiche. Prima della cancellazione viene richiesta una conferma all'Amministratore.

Attori coinvolti:

Amministratore, Sistema.

Trigger:

Amministratore elimina il campo.

Postcondizione:

Il campo viene eliminato dall'elenco di quelli disponibili e non sarà più visibile a Utenti e Organizzatori.

Procedimento:

1. L'amministratore effettua il Login e l'eliminazione
2. Il sistema richiede due volte conferma prima dell'eliminazione
3. Se l'Amministratore procede il campo viene rimosso dall'elenco di quelli disponibili.

2.4.4 UC6.4: Modifica campi

Descrizione:

L'amministratore ha la possibilità di modificare le informazioni relative ad un campo come:

Attori coinvolti:

Amministratore, Sistema.

Trigger:

L'amministratore manda il comando per modificare il campo.

Postcondizione:

Le informazioni relative a un campo vengono modificate e aggiornate per tutti gli utenti che accedono alla piattaforma

Procedimento:

1. L'amministratore effettua il Login e modifica un campo
2. Il Sistema mostra le informazioni modificabili del campo e le rende accessibili all'Amministratore
3. L'amministratore modifica le informazioni
4. L'amministratore conferma la modifica
5. Le informazioni sul campo vengono aggiornate per tutti gli utenti

2.5 UC7: Gestione eventi

2.5.1 UC7.1: Visualizza eventi

Descrizione:

Gli Utenti e l'Organizzatore possono accedere alla visualizzazione degli eventi solo dopo aver effettuato il Login. La lista di eventi disponibili può contenere impianti e campi in luoghi diversi.

Attori coinvolti:

Utente, Organizzatore, Sistema.

Trigger:

Si apre una lista di eventi

Postcondizione:

Viene mostrato l'elenco degli eventi, eventualmente filtrato attraverso qualche parametro.

Procedimento:

1. L'Utente effettua il Login alla piattaforma e si sposta nella visualizzazione dell'elenco degli eventi.
2. L'utente può scegliere se selezionare dei filtri per la visualizzazione degli eventi nell'elenco
3. Il Sistema mostra la lista all'utente nell'ordine desiderato

2.5.2 UC7.2 Inserisci evento

Descrizione:

Gli Organizzatori hanno la possibilità di creare un nuovo evento, selezionando vari parametri al momento della creazione

Attori coinvolti:

Organizzatore, Sistema.

Trigger:

L'Organizzatore manda il comando per la creazione di un evento

Postcondizione:

Viene creato un nuovo evento a cui gli Utenti possono scegliere di partecipare

Procedimento:

1. L'Organizzatore effettua il Login e manda il comando per la creazione di un evento
2. Il Sistema mostra una serie di campi che l'Organizzatore deve compilare per procedere nella creazione
3. L'Organizzatore compila i campi richiesti
4. L'Organizzatore conferma la creazione dell'evento
5. L'evento viene aggiunto all'elenco degli eventi e reso visibile a tutti gli Utenti.

2.5.3 UC7.3: Cancella evento

Descrizione:

L'Organizzatore può decidere di eliminare un evento, rispettando i limiti imposti dal Sistema (non si può eliminare un evento all'ultimo momento)

Attori coinvolti:

Organizzatore, Sistema.

Trigger:

L'Organizzatore elimina l'evento

Postcondizione:

L'evento viene rimosso dalla lista degli eventi nella piattaforma

Procedimento:

1. L'Organizzatore effettua il Login ed elimina l'evento già creato in precedenza
2. Il Sistema richiede due volte la conferma prima di procedere, oppure comunica che non è possibile eliminare l'evento per qualche motivo.
3. Se l'evento viene eliminato, si manda una comunicazione a tutti gli Utenti che si erano iscritti regolarmente
4. L'evento viene rimosso dalla lista ed eliminato definitivamente

2.5.4 UC7.4: Modifica evento

Descrizione:

L'Organizzatore può modificare le informazioni o le regole relative a un evento, come l'orario, la data, l'impianto o il numero di partecipanti richiesto

Attori coinvolti:

Organizzatore, Sistema

Trigger:

L'Organizzatore modifica un evento da lui creato

Postcondizione:

Le informazioni sull'evento vengono modificate e aggiornate, e tutti gli iscritti vengono informati

Procedimento:

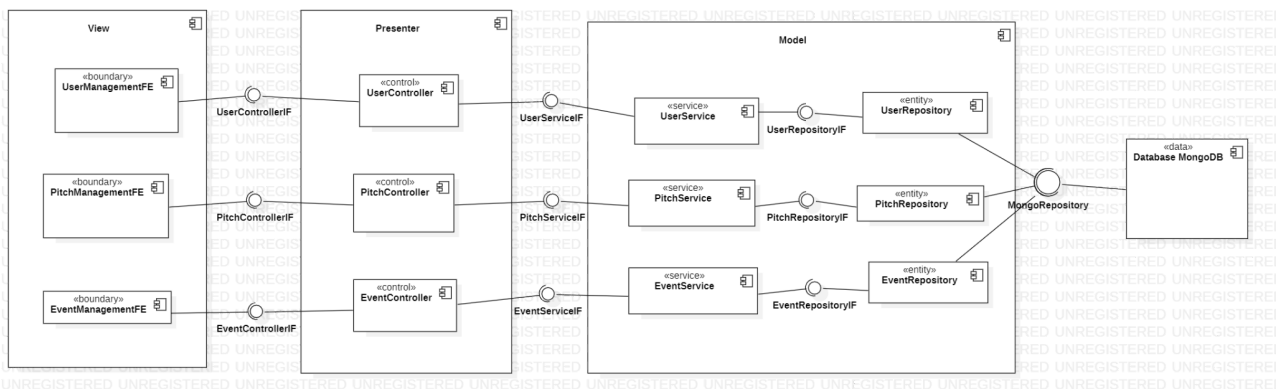
1. L'Organizzatore effettua il Login e modifica un evento da lui creato
2. Il Sistema rende accessibili i campi modificabili all'Organizzatore
3. L'Organizzatore effettua le modifiche e le salva
4. Le informazioni aggiornate sull'evento vengono comunicate a tutti gli iscritti tramite una mail inviata dal Sistema
5. Le informazioni sull'evento sono aggiornate anche sulla lista degli eventi disponibili

2.6 UML Component Diagram

Dai casi d'uso selezionati per questa iterazione, è stato realizzato un diagramma dei componenti del sistema sviluppato sino a questo momento.

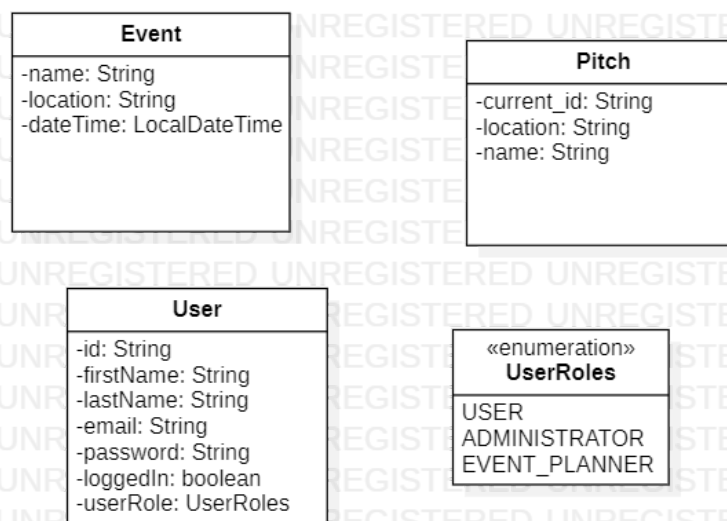
Sono presenti tre tipi di componente:

- boundary - rappresenta i componenti del lato in front-end con cui l'utente e gli amministratori si interfacciano direttamente
- control - rappresenta la business logic del sistema che si occupa di fornire le API ai componenti boundary e gestisce lato back-end i dati provenienti dal database.
- data - componenti che rappresentano dei dati o strutture dati all'interno del database

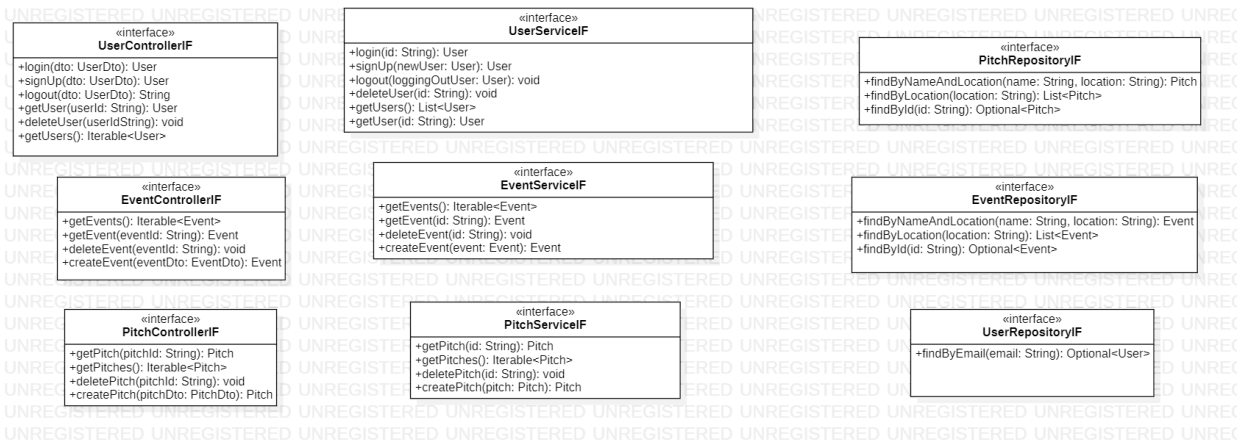


2.7 UML Class Diagram per interfacce e per tipo di dato

I diagrammi sottostanti mostrano le classi implementate in questa iterazione dal punto di vista delle interfacce e delle classi.



Class Diagram per tipo di dato



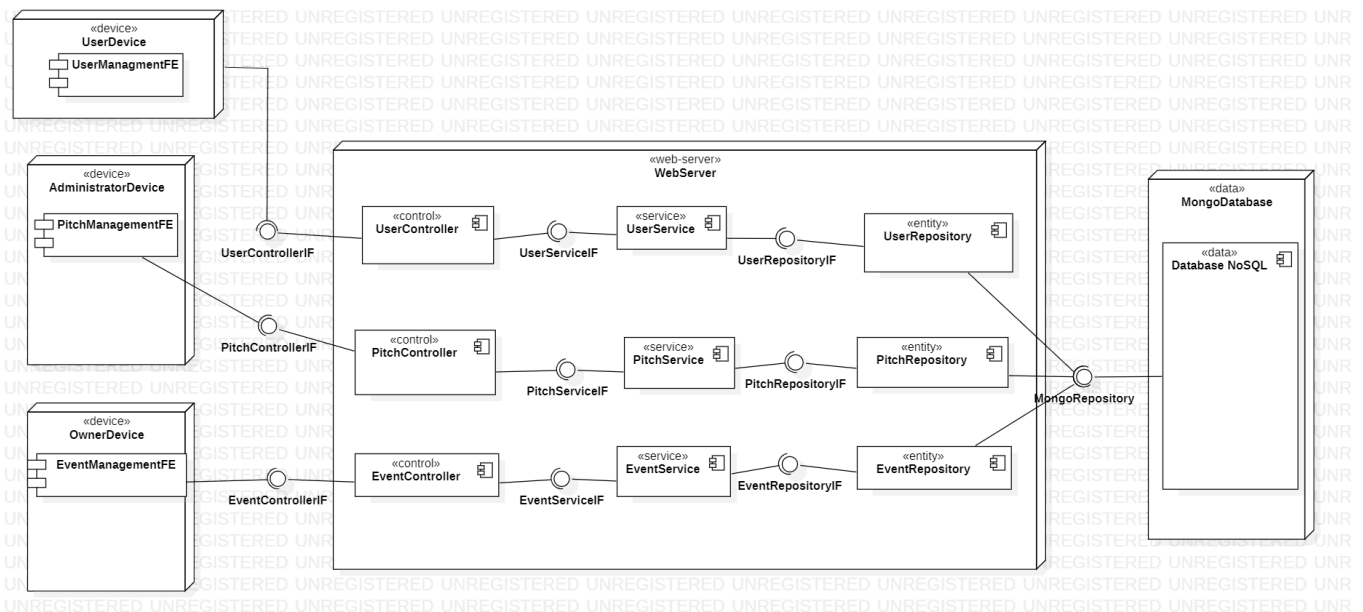
Class Diagram per interfaccia

2.8 UML Deployment Diagram

Tramite un Deployment diagram viene mostrata su quali supporti risiederanno le varie componenti (hardware e software).

Di seguito sono elencati i nodi di deploy:

- Cellulare dell'utente, nodo da cui l'utente è in grado di registrarsi, fare login/logout dalle proprie informazioni personali e prenota partite ed eventi
- Dispositivo Amministratore, nodo da cui l'amministratore che presiede una struttura può fare login/logout può rendere disponibili o meno i campi da gioco
- Dispositivo del Pianificatore di eventi, nodo da cui l'owner può fare login/logout e può bandire eventi.
- Webserver, su cui vengono esposte le API REST
- Database di tipo NoSQL, su cui vengono salvati tutti i dati relativi al sistema del centro sportivo



2.9 Test

2.9.1 Documentazione Test API

Nella corrente iterazione sono stati creati dei test con Postman al fine di verificare il corretto funzionamento delle APIs.

In particolare sono state testate le parti più critiche per quanto riguarda la gestione degli utenti:

- Registrazione dell'utente
- Login/Logout con credenziali dell'Utente
- Stampa di tutti gli Utenti sul database
- Cancellazione dell'utente
- Cancellazione di un utente non esistente
- Registrazione con credenziali già presenti nel database

Qui sotto è riportato una immagine che mostra il funzionamento delle API che svolgono il lavoro sopra descritto:

RUN SUMMARY			1
▼	POST	SignUp Utente	1 0
		Pass Status code is 200	
▶	GET	StampaUtenti	1 0
▶	GET	StampaUtente	1 0
▼	POST	Login	1 0
		Pass Status code is 200	
▼	POST	Logout	1 0
		Pass Status code is 200	
▼	DELETE	DeleteUtenteEsistente	1 0
		Pass Status code is 200	
▼	DELETE	DeleteUtenteNonEsistente	1 0
		Pass Status code is 404	
▼	POST	SignUpUtenteEsistente	1 0
		Pass Status code is 400	

2.9.2 Analisi Statica

Per l'analisi statica abbiamo scelto di utilizzare il tool CodeMR, dopo averlo importato come plug-in in Eclipse. Tutte le misure riportate di seguito si possono trovare nel progetto ([//codemr/StaticAnalysisIteration1/...](#)).

Gli attributi principali dell'analisi statica che abbiamo deciso di riportare sono:

- Accoppiamento = indica la percentuale di accoppiamento tra due classi, ad esempio controllando se l'attributo di una si riferisce all'altra o se hanno metodi che ritornano parametri in comune.
- Mancanza di coesione = misura quanto bene i metodi di una classe sono relazionati tra loro. Nel nostro caso la maggior parte delle classi ha una percentuale bassa o media di mancanza di coesione.
- Complessità = Implica una maggiore difficoltà nell'interpretare l'interazione tra un certo numero di entità. Più è basso meglio è, altrimenti difficile da mantenere.
- Dimensione = misura il numero di metodi o le linee di codice presenti in una classe, se è troppo elevata significa che è possibile dividere la classe in una o più sottoclassi.
- C3 = esprime nell'insieme le tre misure di complessità, coesione, e accoppiamento.

Dai grafici di seguito possiamo notare le varie percentuali dei parametri sopra descritti.

Analysis of webServer

General Information

Total lines of code: 362

Number of classes: 23

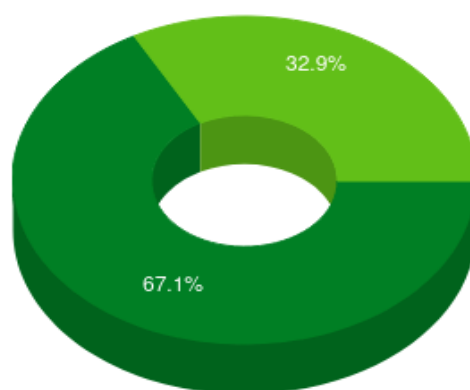
Number of packages: 4

Number of external packages: 15

Number of external classes: 81

Number of problematic classes: 0

Number of highly problematic classes: 0

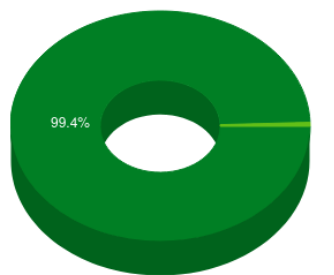


C3

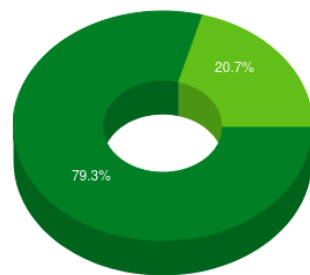
- Very High
- High
- Medium-high
- Low-medium
- Low

Distribution of Quality Attributes

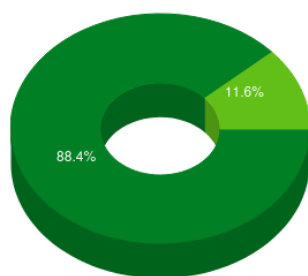
Complexity, Coupling, Cohesion, and Size



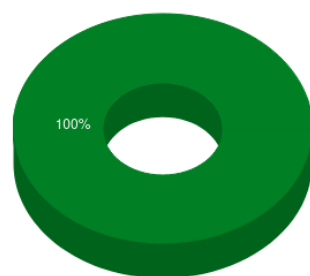
Complexity



Coupling



Lack of Cohesion



Size

Deduciamo da questi dati che vengono rispettate le varie metriche dell'analisi statica.

AMDD: ITERAZIONE 2

In quest'iterazione verranno implementati i seguenti casi d'uso:

- UC5: Gestione delle prenotazioni:
 - UC5.1: Visualizzazione prenotazioni;
 - UC5.2 Inserimento prenotazioni;
 - UC5.3: Cancellazione prenotazioni.
- UC8: Iscrizione evento.
- UC9: Gestione algoritmica delle squadre di un evento.

3.1 UC5: Gestione Prenotazioni

3.1.1 UC5.1: Visualizzazione prenotazioni

Descrizione:

Gli Utenti e l'Organizzatore possono accedere alla visualizzazione delle prenotazioni solo dopo aver effettuato il Login.

Attori coinvolti:

Utente, Organizzatore, Sistema.

Trigger:

Richiesta di visualizzazione di una o più prenotazioni tramite id univoco

Postcondizione:

Viene mostrato l'elenco delle prenotazioni o eventualmente solo la prenotazione legata a uno specifico utente, eventualmente filtrato attraverso qualche parametro.

Procedimento:

1. L'utente/organizzatore effettua il Login alla piattaforma e si sposta nella visualizzazione delle prenotazioni effettuate.
2. L'organizzatore può scegliere se visualizzare una prenotazione specifica oppure se vedere tutte le prenotazioni effettuate. L'utente visualizza solo le proprie prenotazioni
3. Il Sistema mostra ciò che viene richiesto all'utente organizzatore.

3.1.2 UC5.2 Inserimento prenotazione

Descrizione:

Gli utenti hanno la possibilità di creare una prenotazione per un campo che non è già stato prenotato da un altro utente nella stessa ora e nello stesso impianto.

Attori coinvolti:

Utente, Sistema.

Trigger:

L'Utente manda il comando per la creazione di una prenotazione specificando orario, data e impianto.

Postcondizione:

Viene creato una nuova prenotazione e rende il campo non più disponibile per una successiva prenotazione, oppure viene rifiutata.

Procedimento:

1. L'utente effettua il Login e manda il comando per la creazione di una prenotazione
2. Il Sistema ricerca nel database se già esiste una prenotazione per quell'orario
 - 2.1. Se non esiste una prenotazione, crea una nuova prenotazione e viene aggiunta nel database rendendo il campo non più disponibile per quell'ora.

- 2.2. Se esiste già una prenotazione per quel campo alla stessa ora e giorno, viene rifiutata.

3.1.3 UC5.3: Cancellazione prenotazione

Descrizione:

L'utente/organizzatore può decidere di eliminare una prenotazione.

Attori coinvolti:

Utente, Organizzatore, Sistema.

Trigger:

L'Organizzatore/utente richiede l'eliminazione della prenotazione tramite id

Postcondizione:

La prenotazione viene rimossa dalla lista delle prenotazioni nella piattaforma oppure restituisce un messaggio di errore.

Procedimento:

1. L'Organizzatore/utente effettua il Login ed elimina l'evento già creato in precedenza
2. Il Sistema ricerca nella lista delle prenotazioni nel database, la specifica prenotazione
 - 2.1. Se non esiste una prenotazione con tale id, viene restituito un messaggio di errore.
 - 2.2. Se esiste una prenotazione con quell'id, questa viene eliminata dal database

3.2 UC8: Iscrizione evento

Descrizione:

L'utente fa richiesta di iscrizione ad un evento creato da un organizzatore.

Attori coinvolti:

Utente, Sistema.

Trigger:

L'utente fa richiesta al sistema di iscriversi ad un evento tramite un codice id

Postcondizione:

L'iscrizione viene effettuata con successo e viene salvata all'interno del database.

L'iscrizione viene respinta perché supera la dimensione massima delle squadre dell'evento.

L'iscrizione viene respinta perché non ci sono abbastanza posti disponibili.

Procedimento:

1. L'utente effettua il Login e fa richiesta di iscrizione all'evento tramite identificativo
2. Il sistema controlla che l'evento esista, se non esiste viene respinta l'iscrizione
3. Il sistema controlla che il numero di posti disponibili rimanenti sia superiore al numero di componenti nell'iscrizione corrente e che il numero di componenti nell'iscrizione corrente sia inferiore alla dimensione delle squadre specificate dall'organizzatore nell'evento.
4. Se tutti i controlli sono superati, viene effettuato l'iscrizione e il sistema riduce il numero di posti disponibili per quell'evento.

3.3 UC9: Gestione algoritmica della squadra di un evento

Descrizione:

Si tratta di un algoritmo che, sulla base del numero di iscritti per ogni prenotazione dell'evento, crea le squadre della dimensione stabilita per l'evento, mettendo insieme più utenti iscritti di prenotazioni diverse.

Attori Coinvolti:

Organizzatore, Sistema.

Trigger:

L'organizzatore decide di far partire l'algoritmo per la formazione delle squadre per lo specifico evento identificato tramite codice id.

PostCondizione:

Nessuna post-condizione particolare, volendo l'algoritmo può essere eseguito più di una volta e in qualsiasi momento.

Procedimento:

1. L'organizzatore fa richiesta al sistema di formare le squadre per l'evento;
2. il sistema esegue l'algoritmo provando a creare le squadre della dimensione esatta richiesta dall'evento;
3. In caso non riesca a creare le squadre di dimensione esatta, viene rilassata la dimensione richiesta di un numero di giocatori specificato dall'organizzatore nel momento dell'esecuzione dell'algoritmo;
4. Il sistema restituisce le squadre formate specificando per ogni squadra quali prenotazioni ne fanno parte e restituisce anche eventuali prenotazioni che non sono state inserite in una squadra per impossibilità di accoppiamenti.

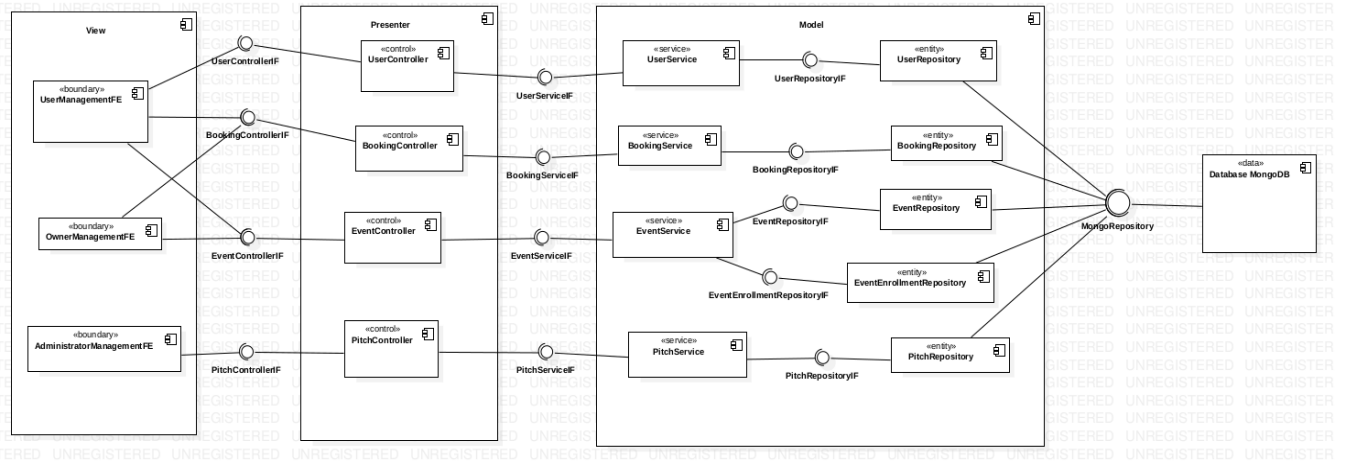
Passi dell'algoritmo

- 1) Il primo passo dell'algoritmo viene attivato nel momento in cui viene fatta una chiamata al relativo endpoint specificando l'id del relativo evento.
- 2) Tramite l'id evento vengono ricercate nel database le prenotazioni e viene restituita la lista (L). Queste contengono, tra gli altri, un campo che specifica quante persone sono iscritte tramite quella singola prenotazione .
- 3) Viene calcolato il numero totale (N) delle persone iscritte sommando il numero di persone specificato in ogni singola prenotazione del particolare evento .
 - a) se $N < \text{MIN_PLAYERS}$ l'algoritmo non calcola la soluzione perchè fino a quel momento il numero totale di iscritti non è sufficiente a far partire l'evento.
 - b) altrimenti, l'algoritmo procede con il passo successivo
- 4) La lista delle prenotazioni L viene ordinata in ordine decrescente sulla base numero degli iscritti per prenotazione
- 5) A questo punto per ogni elemento della lista l'algoritmo prova a formare una squadra di dimensione esattamente di MAX_TEAM (eventualmente sommando gli iscritti di più prenotazioni al fine di arrivare al numero esatto di componenti per ogni squadra)
 - a) Se viene creato un team di dimensione corretta, la squadra viene aggiunta alla soluzione finale e il passo viene ripetuto.

- b) Altrimenti, l'algoritmo procede con il passo successivo
-
- 6) per ogni elemento della lista l'algoritmo prova a formare una squadra di dimensione compresa tra MAX_TEAM e $MAX_TEAM + FLEX$ (eventualmente sommando gli iscritti di prenotazioni diverse al fine di arrivare al numero esatto di componenti per ogni squadra)
 - a) Se viene creato un team di dimensione corretta, la squadra viene aggiunta alla soluzione finale e si ripete il passo corrente
 - b) Altrimenti, l'algoritmo procede con il passo successivo.
 - 7) L'algoritmo prova ad aggiungere le prenotazioni pendenti (quelle che non sono state inserite all'interno di una squadra) a delle squadre già formate in modo che la dimensione della squadra non superi $MAX_TEAM + FLEX$.
 - a) Se la squadra che si viene a creare è ammissibile, questa viene aggiunta alla soluzione e si ripete il passo 7)
 - b) altrimenti, si procede con il passo successivo
 - 8) Viene restituita la soluzione che comprende le squadre e le prenotazioni che l'algoritmo non è stato in grado di utilizzare nella creazione delle squadre (vengono restituiti l'insieme gli id delle prenotazioni che fanno parte di una stessa squadra).

3.4 UML Component Diagram

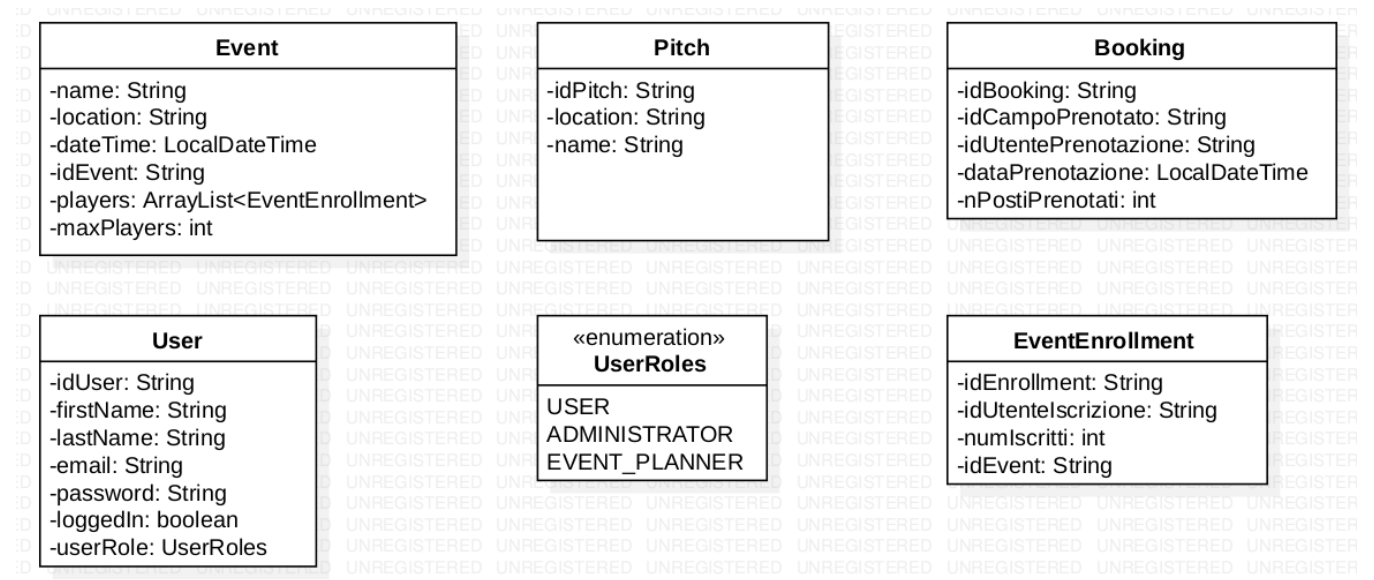
Rispetto all'iterazione precedente è stata aggiunto il componente Booking che permette la gestione di prenotazioni dei campi, oltre alla creazione dell'interfaccia EventEnrollmentRepositoryIF e la relativa entità: EventEnrollmentRepository, in collegamento con EventService.



3.5 UML Class Diagram per i tipi

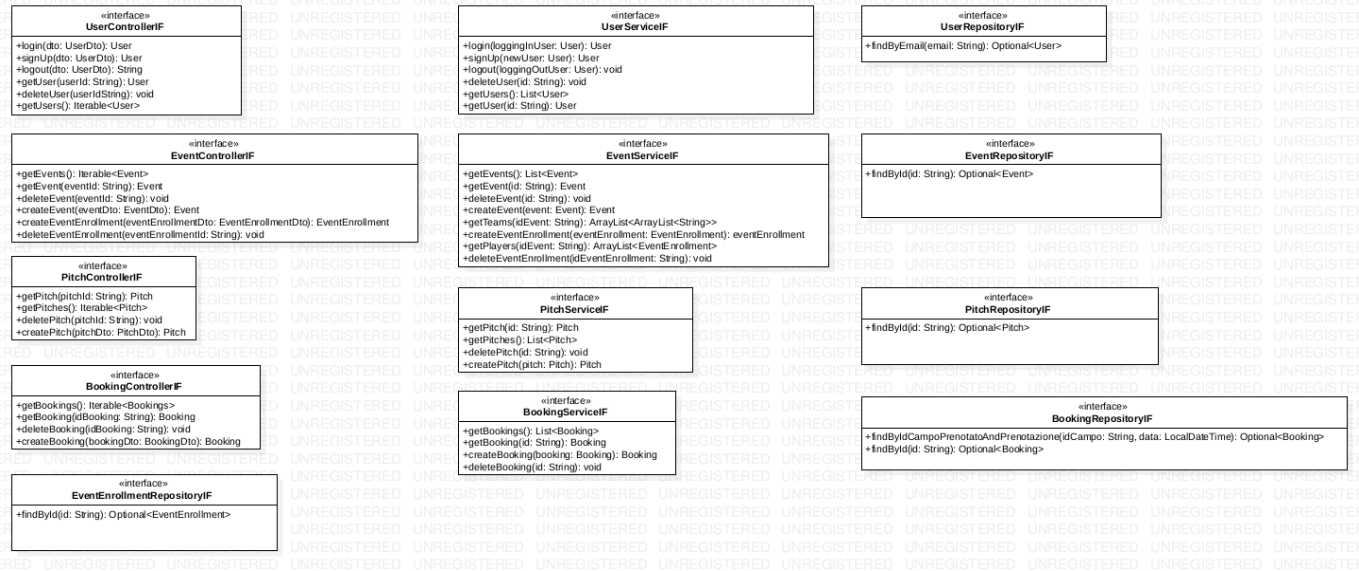
Sono state introdotte due nuove classi:

- Booking: contiene le informazioni che riguardano le prenotazioni di campi.
- EventEnrollment: contiene le informazioni riguardo le iscrizioni a eventi, in particolare il numero di iscritti e i gli id corrispondenti all'Evento e all'Utente che completa l'iscrizione



3.6 UML Class Diagram per le interfacce

Le modifiche più importanti sono state apportate al Class Diagram per le interfacce. In particolare sono stati rimosse operazioni come 'findByLocation' per le interfacce EventRepositoryIF e PitchRepositoryIF. Inoltre sono state aggiunte varie operazioni sulle interfacce già definite all'iterazione precedente, soprattutto nell'interfaccia EventServiceIF, in relazione alla creazione dell'algoritmo. Viene introdotta infine l'interfaccia EventEnrollmentRepositoryIF.

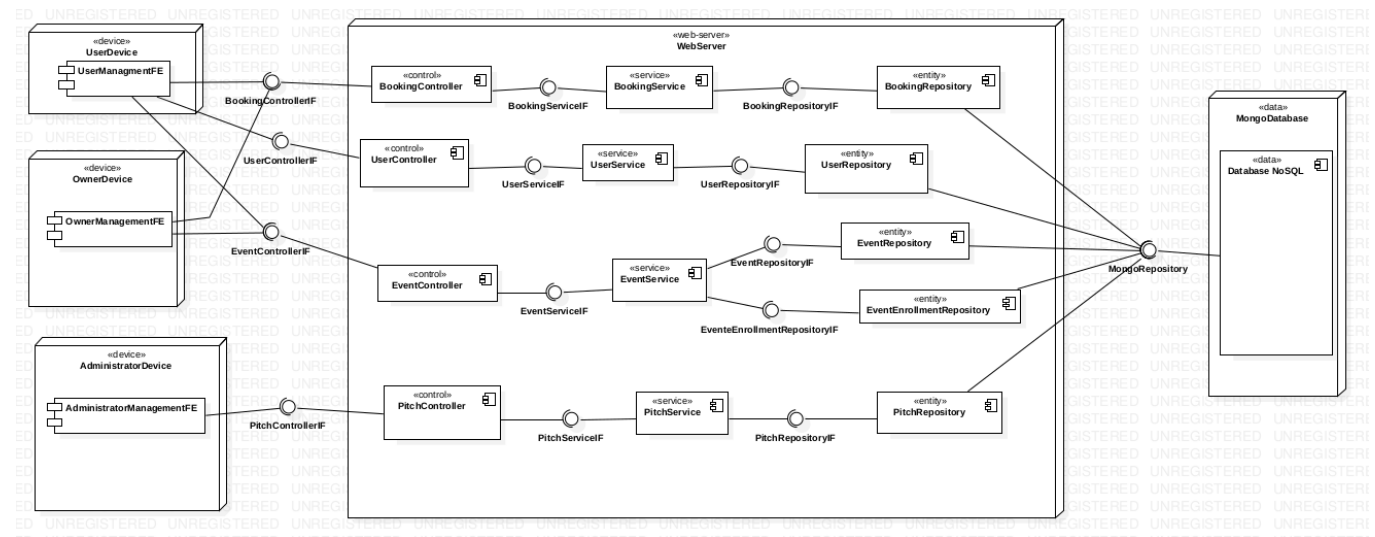


3.8 UML Deployment Diagram

Il nuovo componente chiamato Booking permette l'utilizzo agli User che possono creare ed eliminare prenotazioni.

Anche l'Owner ha accesso a questo componente e in particolare può:

- Creare prenotazioni (in modo da gestire le partite degli eventi).
- Cancellare prenotazioni esistenti (in modo da annullare prenotazioni per vari motivi).



3.9 Testing API

Alla fine di questa iterazione sono stati creati dei test automatizzati per testare la consistenza e il corretto funzionamento delle API.

In particolare sono stati creati dei test per gli endpoint per le seguenti classi:

- Booking: in particolare è stata testato il corretto funzionamento per la creazione, cancellazione e visualizzazione delle prenotazioni
- Event: sono stati testati la creazione, visualizzazione e cancellazione di un evento
- EventEnrollment: iscrizione e cancellazione ad un evento

Inoltre è stato testato l'algoritmo in termini di restituzione di una soluzione (la correttezza è stata testata con la creazione di una caso di test JUnit e commentato nel successivo paragrafo)

RUN SUMMARY

			1
▶	POST	Crea Iscrizione	1 0
▶	POST	Crea Iscrizione Copy	1 0
▶	POST	Crea Iscrizione Copy	1 0
▶	POST	Crea Iscrizione Copy	1 0
▶	POST	Crea Iscrizione Copy	1 0
▶	POST	Crea Iscrizione Copy	1 0
▶	POST	Crea Iscrizione Copy 2	1 0
▶	POST	Crea Iscrizione Copy 3	1 0
▶	POST	Crea Pitch	1 0
▶	POST	creazione book	1 0
▶	GET	visualizza bookings	1 0
▶	POST	Crea Evento	1 0
▶	GET	Crea Squadre (Algoritmo) Copy	1 0
▶	DELETE	Cancella Iscrizione Evento	1 0
▶	DELETE	Cancella Evento	1 0

3.10 Analisi Statica

Abbiamo eseguito l'analisi statica nuovamente tramite CodeMr

Analysis of webServer

General Information

Total lines of code: 699

Number of classes: 33

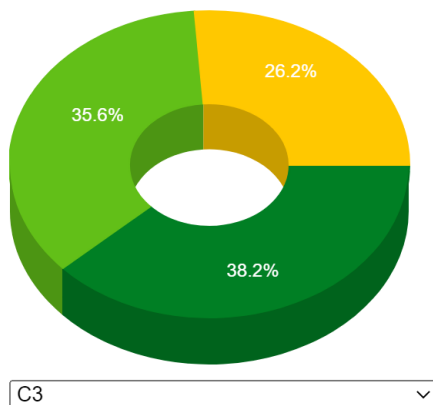
Number of packages: 5

Number of external packages: 15

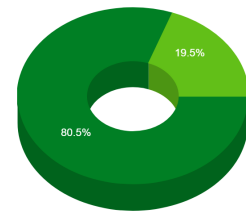
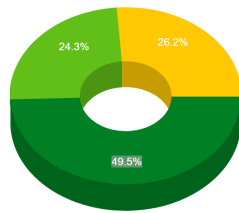
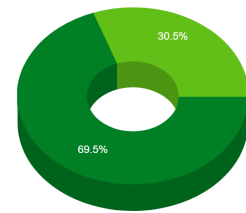
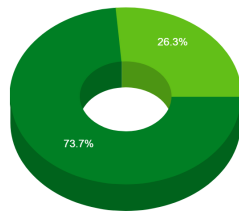
Number of external classes: 115

Number of problematic classes: 0

Number of highly problematic classes: 0



- Very High
- High
- Medium-high
- Low-medium
- Low








Anche in questa iterazione il risultato è soddisfacente. Tuttavia si può notare l'aumento della complessità del software dovuto principalmente alla mancanza di coesione nelle classi dedicate alla gestione di Event.

Questo peggioramento è dovuto all'introduzione di EventEnrollment, ovvero la classe che gestisce la logica dietro le prenotazioni all'evento. Avendo inserito questa parte di codice nelle stesse classi usate da Event sarà più difficile poter cambiare intervenire in fase di manutenzione. Sarà importante in una futura iterazione aumentare la coesione delle classi Event e EventService rimuovendo i riferimenti espliciti a EventEnrollment.

3.11 Analisi Dinamica

L'analisi dinamica è stata svolta tramite test JUnit sull'algoritmo, infatti nel resto del software non vengono svolte azioni particolarmente complesse da richiedere un testing dinamico.

ProvaAlgoritmo		100,0 %	747	0	747
src		100,0 %	747	0	747
(default package)		100,0 %	747	0	747
Main.java		100,0 %	405	0	405
MainTest.java		100,0 %	342	0	342

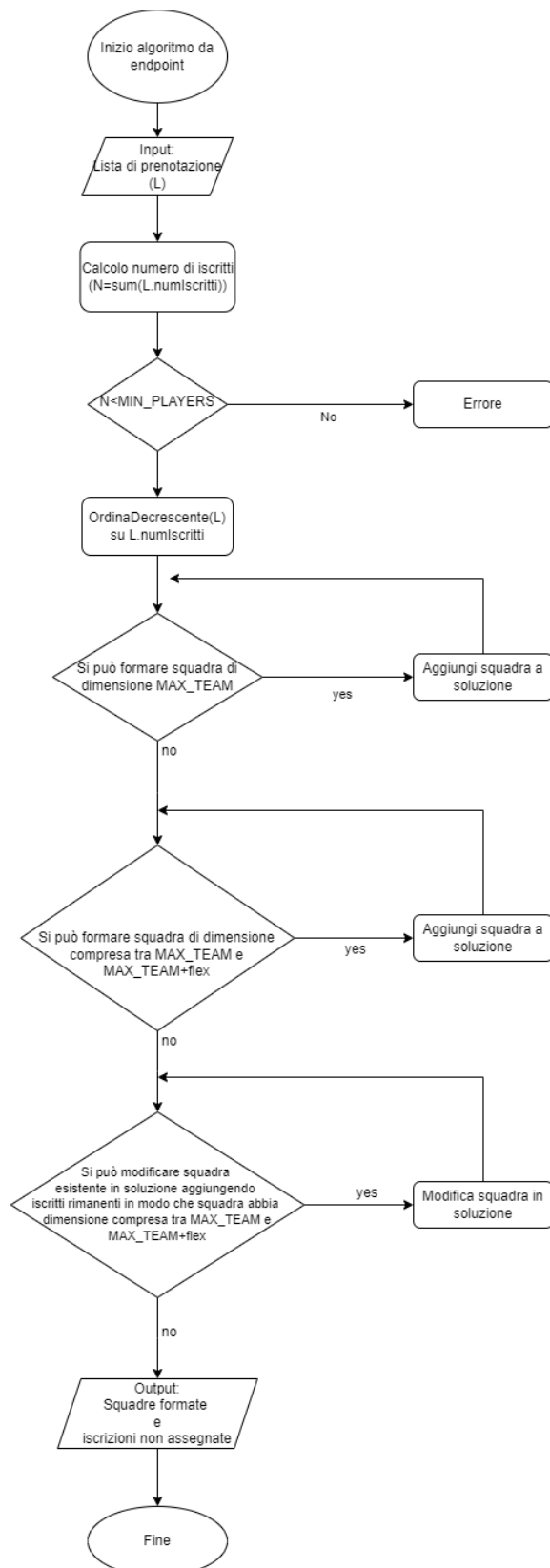
Il test copre il 100% dell'algoritmo andando a simulare tutte le varie casistiche riscontrabili dal software.

In particolare:

- Test1: Input(5). Si conferma che il software assegna l'iscrizione ad una squadra
- Test2: Input(4,1). Il test conferma che l'algoritmo assegni le due iscrizioni ad una squadra.
- Test3: Input(2,2,1). Anche in questo caso il software si comporta correttamente unendo le tre prenotazioni in una squadra.
- Test4: Input(2,2,2). Il codice utilizza la possibilità di formare squadre di dimensione flessibile per creare una squadra composta da 6 giocatori.
- Test5: Input(3,2,1). L'algoritmo prima forma una squadra perfetta da 5 giocatori. Infine nota che può aggiungere l'ultimo giocatore alla squadra, e lo fa.
- Test6: Input(3,3,2). In questo caso si riesce a formare una squadra perfetta da 5 giocatori. Avanza però una prenotazione da 3 giocatori che non riuscirà ad essere inserita nella squadra. Il software restituirà questa prenotazione nell'ultimo elemento del risultato, dove vengono inserite le prenotazioni che non è stato possibile assegnare ad una squadra.

3.12 Analisi algoritmo

3.12.1 Flowchart



3.12.2 Pseudocodice

```
Alg getTeams(String idEvent) : List<List<String>>

List<List<Integer>> result
List<Integer> idx
List<EventEnrollment> list = getListFromDB(idEvent)
List<EventEnrollment> help = list

int nPlayers = 0;

For elem in list
    nPlayers += elem.getNumIscritti()  $O(n)$ 

if nPlayers < MIN_PLAYERS
    throw Exception

OrdinaDecr(list.numIscritti)  $O(n \log n)$ 
```

In questa prima parte di codice:

- Si inizializzano le strutture dati utilizzate per produrre il risultato
- Si contano il numero totale di giocatori iscritti e si controlla che siano sufficienti
- Si ordina la lista di prenotazioni in modo decrescente rispetto al numero di giocatori.

In questa parte di codice la complessità è $(n \log n)$ ed è data dall'ordinamento supponendo di usare un MergeSort.

```
For i in help
    idx.add(i)
    int sum = help[i].getNumIscritti()

    if sum == MAX_TEAM
        result.add(idx)  $O(n/2)$ 

    For j=i+1 in help
        if sum + help[j].getNumIscritti() == MAX_TEAM
            sum += help[j].getNumIscritti()
            idx.add(j)
            result.add(idx)
        else if sum + help[j].getNumIscritti() < MAX_TEAM
            sum += help[j].getNumIscritti()
            idx.add(j)

    if sum == MAX_TEAMS  $O(MAX\_VALUE)$ 
        For x in idx
            help[idx[x]] = MAX_VALUE;

idx.clear();
```

$O(n * (n/2 + MAX_VALUE)) = O(n^2)$

Nella prima scansione per la formazione di squadre abbiamo due cicli for annidati in un ulteriore loop.

Ciò introduce una complessità pari a $O(n(\frac{n}{2} + MAX_VALUE))$ che nel calcolo asintotico può essere scritta come $O(n^2)$

```
// inizio algoritmo flessibile
```

```

For i in help
  idx.add(i)
  int sum = help[i].getNumIscritti()

  For j=i+1 in help
    if sum + help[j].getNumIscritti() > MAX_TEAM && <= MAX_TEAM + FLEX
      sum += help[j].getNumIscritti()
      idx.add(j)
      result.add(idx)
    else if sum + help[j].getNumIscritti() < MAX_TEAM
      sum += help[j].getNumIscritti()
      idx.add(j)

  if sum > MAX_TEAM && <= MAX_TEAM + FLEX
    For x in idx
      help[idx[x]] = MAX_VALUE;
    idx.clear();

```

$O(n*(n/2 + MAX_VALUE)) = O(n^2)$

$O(n/2)$

$O(MAX_VALUE)$

La seconda scansione è esattamente uguale alla prima per quanto riguarda la complessità, varia solamente il vincolo per l'aggiunta di una squadra. In questo caso è permessa maggior flessibilità nella creazione delle squadre.

```
//aggiunta di giocatori in squadre esistenti
```

```

For i in help
  if help[i].getNumIscritti() < MAX_TEAM
    For j in result
      int sumSquad = 0
      For z in result[j]
        // stiamo controllando da quanti membri è composta la j-esima
        squadra
        sumSquad += lista[result[j][z]].getNumIscritti();
      if sumSquad + help[i].getNumIscritti() < MAX_TEAM + FLEX
        result[j].add(i)
        help[i] = MAX_VALUE

```

$O(n*n) = O(n^2)$

$O(n*MAX_VALUE) = O(n)$

$O(MAX_VALUE)$

In quest'ultima parte dell'algoritmo si cerca di aggiungere gli ultimi partecipanti rimasti a squadre già esistenti. Per fare ciò si implementano 3 cicli for annidati, in cui però quello più interno non ha una complessità dipendente da n.

Infatti in questo ciclo (in verde nell'immagine) si ispezionano le iscrizioni di una squadra che, nel caso peggiore in cui tutte le prenotazioni sono composte da un solo giocatore, sono pari a MAX_VALUE.

Si ottiene quindi una complessità totale pari a $O(n * n * MAXVALUE)$ che con il calcolo asintotico può essere scritta come $O(n^2)$.

In conclusione possiamo quindi dire che la complessità totale dell'algoritmo è $O(n \log n + n^2 + n^2 + n^2)$ che asintoticamente è pari a $O(n^2)$.