A picture containing logo

Description automatically generated

DOCUMENTAZIONE

Componenti del gruppo:

Carminati Matteo n°matricola: 1066354

Torri Lorenzo n°matricola: 1069047

Università degli Studi di Bergamo - Corso di Ingegneria del Software – a.a. 2021/2022

**CAPITOLO 3 – Software Life Cycle**

La scelta del modello di vita del nostro software è ricaduta su uno di tipo agile, in particolare XP. Questa scelta è legata all’organizzazione del team che è composto esclusivamente da due persone, inoltre c’è la necessità di apportare al progetto rapidi e funzionali cambiamenti.

Extreme Programming è ottimale per il team in quanto la gestione del lavoro si basa interamente sul pair programming. I due componenti del team infatti lavorano per la maggior parte del tempo insieme riuniti in calls di Microsoft Teams. Entrambi conoscono perfettamente ogni aspetto del progetto sviluppato. Ogni componente del team è in grado di cambiare il codice, in quanto lo conosce e lo sa manipolare correttamente.

Per lo sviluppo del codice i due sviluppatori hanno deciso di dividersi il lavoro, tuttavia le modifiche saranno sempre sottoposte a verifica da parte dell’altro membro. Tali verifiche saranno effettuate durante brevi riunioni da remoto in modo che si possa avere un confronto diretto e immediato.

L’idea è quella di iniziare a lavorare su una prima versione basilare ma funzionante dell’applicazione, per poi rilasciare costantemente delle piccole versioni del sistema che vadano a migliorare la situazione di partenza e che permettano di verificare le funzionalità implementate.

Al termine dell’introduzione di una nuova funzionalità all’interno della applicazione bisognerà fare alcuni test. La fase di testing ha un ruolo chiave e verrà affrontata con più precisione in seguito (capitolo 13).

Il team non si focalizzerà sulla parte grafica, terrà infatti un design grafico che sia il più semplice possibile, proprio perché quella commissionata è solo un prototipo della vera applicazione che potrà essere implementata in futuro.

Come ultima precisazione il team predilige, come già specificato nel Project Plan al punto 8, un approccio di tipo Model driven architecture, dunque, quando possibile, i cambiamenti dell’applicativo vengono effettuati sui modelli UML implementati e non direttamente sul codice.

La software life cycle è stata parzialmente implementata da parte degli sviluppatori anche mediante formalizzazione eseguita con il diagramma statechart del tool UML.

Graphical user interface

Description automatically generated

**CAPITOLO 5 – People Management and team Organization**

L'organizzazione del team e la gestione delle persone che ne fanno parte è riconducibile perlopiù all'approccio agile, i due sviluppatori infatti collaborano in ogni aspetto del loro lavoro, applicando anche pair programming.

Per certi aspetti la gestione dei membri del team viene affrontata seguendo lo schema proposto da Mintzberg; in particolare vi sono alcune analogie con il meccanismo di coordinazione di adhocracy. L’organizzazione del lavoro avviene infatti prediligendo meeting e momenti di scambio di idee tra i due membri del team.

Altro aspetto tipico di adhocracy, e ripreso dal team di sviluppo, è la scelta di coordinare il lavoro attraverso mutual adjustment che i membri svolgono con molta regolarità.

Tali correzioni avvengono però tenendo sempre presente quelli che sono gli obiettivi prefissati, ponendo il focus sul prodotto finale considerato nella sua completezza e mai suddiviso in parti.

Per quanto riguarda la strutturazione del team di lavoro, questa si discosta da adhocracy e si addice molto di più ad un modello agile; all’interno del team infatti non esiste alcun tipo di gerarchia o di rigida suddivisione dei compiti.

Questo approccio è stato ritenuto quello più adatto poiché i due componenti del team hanno percorsi formativi e skills molto simili, pertanto svolgono compiti analoghi e godono entrambi di grande autonomia non dovendo rispondere alle direttive di un superiore.

Lo svolgimento del lavoro segue sempre gli standard legati ai linguaggi utilizzati.

Il team ha solo specifiche indicative riguardanti l’ammontare di ore di lavoro richieste per lo sviluppo della piattaforma.

Per poter organizzare meglio il workflow gli sviluppato hanno deciso di utilizzare il servizio Kanban integrato a Github che permette, con le sue boards, di visualizzare e scegliere gli obiettivi di giornata.

**CAPITOLO 6 - Software Quality**

Il cliente non ha specificato dei requisiti di qualità relativi alla piattaforma richiesta; pertanto, questi sono stati determinati direttamente dal team di lavoro.

In particolare i due sviluppatori hanno deciso di seguire le direttive del modello McCall per avere dei criteri qualitativi da rispettare durante il lavoro svolto.

Alcuni parametri qualitativi saranno valutati in fase di sviluppo, altri necessariamente verranno presi in considerazione solo al termine dell’applicativo, o comunque durante le fasi finali di sviluppo.

Si riportano i fattori di qualità perseguiti dal team e classificati secondo il modello McCall:

* Product operation
  + Correctness

Il team vuole garantire un applicativo correttamente funzionante e che rispetti i requisiti riposti

* + Reliability

Per la reliability si rimanda alle considerazioni fatte durante la fase di Software Testing del CAPITOLO 13

* + Efficiency

La piattaforma non richiede particolari risorse, per cui ci si aspetta che possa performare in maniera uguale su ogni dispositivo, indipendetemente dall’hardware di quest’ultimo.

* + Integrity

Non ci sono criticità particolari nell’ambito della sicurezza.

* + Usability

La piattaforma, per poter essere eseguita, necessita di una JVM installa nel dispositivo.

* Product revision
  + Maintainability

Per la reliability si rimanda alle considerazioni fatte durante la fase di Software Testing del CAPITOLO 14

* + Testability

Per la reliability si rimanda alle considerazioni fatte durante la fase di Software Testing del CAPITOLO 13

* Product transition
  + Portability

E’ possibile utilizzare la piattaforma su qualsiasi dispositivo che abbia installata una JVM.

**CAPITOLO 9 – Requirements Engineering**

Come spiegato già all’interno del project plan l’applicativo sviluppato è al momento solo un prototipo di quella che potrebbe essere la piattaforma di prenotazione online definitiva e commercializzabile.

Il lavoro non è stato commissionato dal Presidente dell’associazione, ma è una proposta che i due sviluppatori hanno deciso di presentare. Pertanto non sono stati forniti dal Presidente requisiti particolari inerenti alla piattaforma stessa. Il team tuttavia, basandosi anche sull’esperienza dei singoli membri nell’ambito delle prenotazioni di strutture sportive, ha provveduto a stilare una serie di requisiti che vorrebbero vedere implementati in una piattaforma di questo genere.

I requisiti nella fase di elicitation sono stati organizzati in ordine gerarchico seguendo il modello Moscow:

* Must have
  + L’utente ha la possibilità di scegliere lo sport che vuole praticare.
  + L’utente ha la possibilità di scegliere l’orario di prenotazione.
  + L’utente ha la possibilità di scegliere la struttura che vuole prenotare.
  + La piattaforma cerca strutture con le indicazioni dell’utente.
  + L’utente conferma la prenotazione.
  + La piattaforma calcola il costo della prenotazione.
* Should have
  + L’utente ha la possibilità di scegliere se desidera avere lo spogliatoio o no.
  + L’utente ha la possibilità di effettuare prenotazioni singole oppure collettive.
  + L’utente ha la possibilità di indicare una lista di amici se la sua prenotazione è collettiva.
  + Le strutture già registrate possono registrare nuovi campi.
  + Le strutture già registrate possono registrare nuovi spogliatoi.
  + Strutture non ancora registrate sulla piattaforma possono registrarsi.
  + La piattaforma distingue tra clienti standard e clienti convenzionati.
  + La piattaforma calcola il costo della prenotazione rispetto alla tipologia di cliente.
  + La piattaforma calcola il costo a persona qualora la prenotazione fosse collettiva.
* Could have
  + La piattaforma, qualora il campo non sia disponibile, propone una serie di strutture disponibili e che rispettano i canoni indicati precedentemente dall’utente.
  + La piattaforma, qualora il campo non sia disponibile, propone gli orari in cui il campo risulta essere ancora disponibile per il giorno specificato dall’utente.
  + L’utente ha la possibilità di scegliere tra le strutture proposte in automatico dalla piattaforma.
  + L’utente ha la possibilità di disdire prenotazioni già effettuate.
  + Le strutture non registrate sulla piattaforma hanno la possibilità di registrarsi inserendo i loro campi e spogliatoi.
* Won’t have
  + La piattaforma informa l’utente se la struttura fornisce al cliente materiale tecnico oppure no.
  + Le strutture possono bandire utenti sgraditi.
  + Interfaccia grafica curata

Nonostante il team non abbia adottato un modello di organizzazione di tipo document-driven ha deciso di riferirsi allo standard IEEE 830 per la specifica dei requisiti al fine di definirli in maniera più ordinata e precisa.

1. *Introduction*
   1. *Purpose*

Questo documento riporta i requisiti definiti per la realizzazione di un prototipo di una piattaforma di prenotazioni online di attività sportive. Il documento costituisce una linea guida per il successivo sviluppo dell’applicativo stesso.

* 1. *Scope*

Questo prototipo vuole creare una prima simulazione semplificata di quella che potrebbe essere una piattaforma più completa ed elaborata, che possa permettere agli utenti frequentatori di centri sportivi di poter prenotare con facilità i campi adibiti a tale scopo. Ogni struttura possiede dei campi e degli spogliatoi, inoltre le strutture hanno la possibilità di aggiungere nella piattaforma eventuali nuovi campi e spogliatoi.

Le strutture possono impedire che utenti indesiderati possano effettuare prenotazioni presso i loro campi. La piattaforma ha come obiettivo futuro quello di riunire il maggior numero possibile di strutture sportive della provincia di Bergamo, così da poter offrire un servizio migliore.

* 1. *Definition, acronyms and abbreviations*

**Utente standard**: cliente frequentatore di centri sportivi che vuole ricercare un campo ed effettuare una prenotazione.

**Utente convenzionato**: cliente uguale a quello standard, ma dispone di uno sconto nel calcolo del costo totale della prenotazione.

**Campi**: impianto di proprietà di una certa struttura e prenotabile dall’utente per praticare un determinato sport.

**Spogliatoio**: stanza di proprietà di una certa struttura messa a disposizione ai clienti per cambiarsi e lavarsi.

**Strutture**: centro sportivo formato da campi e spogliatoi.

**Prenotazione collettiva**: un utente può decidere di fare una prenotazione a suo nome e di indicare una lista di amici con cui praticare l’attività prenotata.

**Ban**: una struttura può decidere di impedire a determinati utenti indesiderati di effettuare prenotazioni a loro nome presso gli impianti gestititi dalla suddetta struttura.

* 1. *Overview*

La sezione 2 serve per dare una visione generale del funzionamento della piattaforma, mentre la sezione 3 da maggiori specifiche riguardo ai requisiti dell’applicativo. All’interno dello standard sono stati riportati solamente i requisiti identificati come must have e should have dal modello Moscow, gli altri requisiti di could e won’t have verranno presi in considerazione solo qualora ci fosse tempo per implementarli.

1. *Overall description*
   1. *Product Perspective*

Il prototipo verrà interamente sviluppato in linguaggio Java. Non avrà nessuna interazione con altri sistemi, dunque tutte le informazioni e i salvataggi legati all’applicativo saranno conservati mediante strutture proposte dal linguaggio Java senza l’ausilio di sistemi di database.

* 1. *Product function*

Il sistema mette a disposizione queste funzionalità

* L’utente può ricercare un campo in una determinata struttura, a un orario stabilito per poter praticare un determinato sport
* L’utente può prenotare oltre a un campo anche uno spogliatoio
* L’utente ha la possibilità di indicare una lista di amici effettuando quindi una prenotazione collettiva
* La piattaforma calcola preventivamente il costo totale della prenotazione basandosi sulla tipologia di utente (convenzionato o non convenzionato) e nel caso fosse una prenotazione collettiva calcolare anche il costo a persona.
* Le strutture possono registrare nuovi spogliatoi e campi
* Le strutture possono controllare le prenotazioni dei loro campi e spogliatoi
  1. *Constraints*

Le strutture per poter entrare nella loro zona dedicata necessitano di un identificativo che viene fornito dall’applicativo nel momento in cui vengono inserite per la prima volta nel sistema.

Il cliente invece per poter effettuare le prenotazioni deve indicare il suo codice fiscale, che verrà verificato dal personale della struttura al momento dell’attività.

1. *Specific requirements*
   1. *Functional requirements*
      1. *Selezione tipologia*

Il sistema propone un menù iniziale da cui ci si identifica come cliente o come struttura. In base alla selezione fatta, il sistema entra nella sezione dedicata. Ad ogni momento è possibile tornare al menù iniziale.

* + 1. *Funzioni utente*
       1. *Ricerca Campo*

Questa funzionalità permette all’utente di ricercare un campo sportivo in un determinato giorno e orario presso una struttura e per praticare un determinato sport.

*3.2.2.2 Prenotazione Campo e Spogliatoio*

Si permette all’utente di prenotare il campo ottenuto dalla funzione ricerca campo utilizzando il proprio codice fiscale. Qualora l’utente fosse interessato anche a utilizzare uno spogliatoio, la piattaforma verifica se la struttura ha a disposizione uno stesso nell’orario specificato e in caso positivo lo prenota.

* + - 1. *Prenotazione collettiva*

Qualora l’utente abbia intenzione di praticare l’attività con amici, può specificarne i nominativi durante la prenotazione.

* + 1. *Funzioni struttura*
       1. *Identificazione Struttura*

Per entrare nella propria area dedicata, una struttura deve superare una fase di identificazione, dove è necessario inserire un identificativo che le è stato fornito dal sistema quando è stata inserita nello stesso.

* + - 1. *Aggiunta Campi e/o Spogliatoi*

Ogni struttura registrata nel sistema ha la possibilità di aggiungere nuovi campi e spogliatoi nel suo elenco dedicato.

* + - 1. *Ban Utenti*

La struttura ha la possibilità di vietare ulteriori prenotazioni da parte di un utente, identificato in maniera univoca dal suo codice fiscale.

* + - 1. *Visualizzare Elenco Prenotazioni*

Ogni struttura ha la possibilità di controllare tutte le prenotazioni effettuate presso i suoi campi e spogliatoi.

* 1. *Performance requirements*

Non esistono particolari limitazioni legate al sistema, se non quelli imposti dalle classi java che verranno impiegate per la realizzazione.

Non ci sono particolari vincoli temporali per quanto riguarda l’esecuzione, l’interesse è unicamente quello di assicurare consistenza dei dati.

* 1. *Software system attributes*
     1. *Security*

Dato che l’applicativo è un prototipo che non viene utilizzato contemporaneamente da più utenti, ma viene esclusivamente eseguito in locale e da un utente alla volta, non necessita di meccanismi di mutua esclusione per la gestione dell’accesso a risorse condivise, in quanto non esistono accessi concorrenti.

**CAPITOLO 10 – Modelling**

Per visualizzare i diagrammi UML si rimanda al file “Diagrammi UML.mdj”.

USE CASE DIAGRAM

Il diagramma mostra gli attori che possono interagire con la Piattaforma e cosa possano fare nella stessa. clienti  iscritti e non iscritti sono generalizzati da cliente.

si può fare anche una prenotazione senza altri partecipanti

CLASS DIAGRAM

Non possono esistere campi o spogliatoi non associati a una struttura, di conseguenza usiamo una composition. Inoltre una prenotazione non può esserci senza un campo, quindi avrà anche in questo caso una composition, mentre nel caso degli spogliatoi, possono esistere anche senza prenotazione e una prenotazione può esistere senza campo.

Non consideriamo un login perchè assumiamo che dato che comunque è un test gli account sono già creati dagli sviluppatori per i vari tester e non c’è temporaneamente la possibilità di aggiungere nuovi iscritti.

Ogni campo e spogliatoio ha associato un prezzo che può essere ottenuto dalla prenotazione mediante calcolaprezzo() che verifica il prezzo in base al campo e allo spogliatoio associati alla prenotazione stessa.

Cliente è astratta perchè ci sono due metodi di calcolo del prezzo della prenotazione in base alla tipologia del cliente: se è standard semplicemente ritorna il prezzo della prenotazione, se è convenzionato, invece, aggiunge uno sconto.

Se creiamo una nuova prenotazione questa deve essere inserita nella lista delle prenotazioni fatte dal cliente.

STATE MACHINE

Nel momento in cui si passa da idle a attiviytà utente allora vuol dire che sta effettuando una prenotazione.

Qui abbiamo un fork perché separiamo la ricerca del campo e dello spogliatoio in modo parallelo nelle due regioni di un orthogonal state, perchè potremmo avere che io prendo sia spogliatoio e campo, oppure solamente il campo. Prima di fare questa scelta in parallelo ho due metodi di entrata dove scelgo lo sport che voglio praticare e scelgo anche la struttura dove vorrei giocare, di modo tale poi da fare una ricerca in parallelo tra gli spogliatoi e i campi di quella struttura.

La prenotazione è composta da una prenotazione del campo e una dello spogliatoio, quella dello spogliatoio non può esistere senza quella del campo. Le due regioni poi si riuniscono all’interno di una conferma della prenotazione.

Per la scelta dello spogliatoio c’è un decision node che in base a una certa variabile che ci spiega se il cliente è intenzionato a prenotare uno spogliatoio o meno. In caso negativo si raggiunge il final state nella regione dell’ortogonal e quindi si procederà con solo la prenotazione del campo che va verso il secondo decision node. In caso affermativo invece i due child del fork rimangono entrambi attivi e si riuniscono dopo verso il secondo decision node e coincide con una prenotazione in una struttura con campo e spogliatoio.

Per poter confermare la prenotaizone è necessario andare a verificare alcuni campi come il fatto che l’orario del campo scelto sia libero. All’inizio settiamo l’orario iniziale ideale per il cliente nel settaggio. Qualora il campo sia occupato o lo spogliatoio se serve sia occupato allora si torna nei settaggi iniziali.

Qualora il cliente non trovi un orario che gli vada bene in quella struttura può decidere di tornare alle condizioni iniziali di settaggio per scegliere una nuova struttura.

Nelle attività di setting la ricerca della struttura c’è una self per trovare la struttura fino a quando non ne troviamo una per cui non siamo bannati. Le altre sono messe in entry perchè non le rifacciamo tutte le volte che torniamo nello stato perchè la self non fa uscire dallo stato, quindi vengono svolte sono una volta e basta. L’unica in do è la ricerca della struttura

**CAPITOLO 11 – Software Architecture**

Il team ha deciso di sviluppare un’architettura del software seguendo lo standard IEEE 1471, per poter suddividere l’applicativo in componenti per poterne semplificare il funzionamento e come modello guida per la realizzazione della piattaforma.

Per l’analisi delle viewpoint si è seguito il modello di Bass, nel quale si suddividono le viewpoint in tre classi: Module Viewpoint, Component and Connector Viewpoint e Allocation Viewpoint.

In particolare il team si è soffermato sulle prime due tipologie.

Il Module Viewpoint è stato utilizzato per rappresentare una vista statica dell’applicativo, il quale è stato diviso in livelli (Layered). In particolare sono stati individuati cinque diversi livelli, dove ogni livello può sfruttare i servizi offerti da quelli sottostanti. La piattaforma è pensata per poter essere utilizzata sia dagli utenti che vogliono effettuare le prenotazioni, sia per le strutture che vogliono invece accedere alla loro area riservata, di conseguenza i due attori faranno un utilizzo differente dell’applicativo, nonostante questo sia il medesimo per entrambi. Al quinto livello sono quindi presenti le due diverse tipologie di utilizzatori della piattaforma. A quello sottostante è presente la Piattaforma che rappresenta un’astrazione del nostro sistema.

Dal terzo livello in poi sono state specificate le principali classi che compongono il sistema, ponendo l’attenzione sulle loro interazioni.

Per quanto riguarda la classe struttura, questa sfrutta i servizi delle classi Campo e Spogliatoio, le quali a loro volta si fondano su quelli proposti dalla classe Prenotazione. La classe Cliente, invece, salta il secondo livello e sfrutta direttamente i servizi proposti dalla classe Prenotazione.

Per approfondire al meglio i servizi offerti dalle singole classi ci si rifà al Class diagram.

Graphical user interface, text, application

Description automatically generated

Gli sviluppatori hanno realizzato anche un Component and Connector Viewpoint, per poter avere una vista dinamica del sistema, individuando i componenti principali dell’applicativo e mostrandone le interazioni.

Si propongono due punti di vista differenti: quello dell’utente e quello della struttura iscritta alla piattaforma.

Per quanto riguarda la vista dell’utente, c’è una funzionalità di Ricerca Struttura che viene erogata dalla Piattaforma. Questa permette di individuare all’interno della piattaforma una struttura, presente tra le strutture del registro, che rispetti i parametri di prenotazione indicati dall’utente.

La vista della struttura, invece, permette di svolgere alcune attività, indicate con Attività Struttura, sempre erogate dalla Piattaforma. In particolare è necessario che prima si effettui un’autentificazione, per poter poi interagire in completa libertà con gli altri componenti del sistema stesso.

Si specifica che il connector che lega Struttura con Cliente vuole indicare la possibilità di una struttura di eseguire un ban nei confronti di un oggetto della classe Cliente.

Graphical user interface, application

Description automatically generated

Graphical user interface, application

Description automatically generated

**CAPITOLO 12 – Software Design**

Per la realizzazione della piattaforma si utilizza il linguaggio di programmazione Java, essendo questo un linguaggio object-oriented, la squadra di lavoro ha deciso di definire la parte di design dell’applicativo attraverso l’object-oriented analysis and design methods (OOAD).

Precisando che non tutti gli oggetti qui riportati faranno parte delle classi effettive che compongono la piattaforma realizzata (ad esempio non esisterà una classe codice fiscale, questo sarà invece un attributo di un’altra classe), di seguito si riporta il modello UML definito individuando gli oggetti del sistema e le loro interazioni.

Immagine che contiene testo, computer, portatile, monitor

Descrizione generata automaticamente

Questo diagramma non permette di definire in maniera chiara la dinamicità del sistema. Ad esempio, il diagramma non permette di specificare il procedimento temporale con cui avviene una prenotazione (settaggio parametri, seguito da controllo disponibilità ed infine conferma o respinta della prenotazione).

Per questo motivo gli sviluppatori rimandano al sequence diagram qualora si volesse avere una migliore visione del funzionamento del sistema.

Le entità identificate nella prima parte della fase di design vengono esplicitate seguendo lo standard del modello IEEE 1016 il quale suddivide ogni entità in una serie di attributi.

|  |  |
| --- | --- |
| **Identification** | Struttura |
| **Type** | Classe |
| **Purpose & Function** | Permette di identificare univocamente le strutture sportive del sistema.  Ha associato una lista di campi e una lista di spogliatoi, entrambe modificabili.  Può specificare utenti indesiderati (ban).  Può consultare le prenotazioni associate ai suoi campi e spogliatoi.  Tiene conto del numero di prenotazioni fatte da ciascun cliente presso i suoi campi. |
| **Dependencies** | Usa le classi Cliente, Spogliatoio e Campo.  È usata dalla classe Registro Strutture. |
| **Interfaces** | Utilizza i costruttori di Campo, Spogliatoio e Cliente qualora si aggiungano nuovi oggetti nelle liste di Struttura indicate in precedenza.  Sfrutta i metodi di Spogliatoio e di Campo per poter stampare le prenotazioni. |
| **Processing** | Fare dopo |
| **Data** | Gli attributi identificatovi della Struttura sono: Nome, Indirizzo, Città.  Ha un attributo IDstruttura generato in automatico dal sistema e che funziona da password.  Possiede le liste: ListaCampi, ListaSpogliatoi, ListaBan.  Possiede una hashmap ConteggioPrenotazioni che associa ad ogni cliente il numero di prenotazioni fatte presso la struttura. |

|  |  |
| --- | --- |
| **Identification** | Campo |
| **Type** | Classe astratta |
| **Purpose & Function** | È una classe astratta perché generalizza le classi specifiche Campo Calcetto, Campo Basket e Campo Tennis.  Riporta solo attributi e metodi comuni a tutte le tipologie di campi.  Permette di aggiungere prenotazioni modificando la lista ListaPrenotazioneCampi associata.  Permette di stampare le prenotazioni associate. |
| **Dependencies** | Usa la classe Prenotazione.  È usata dalla classe Struttura. |
| **Interfaces** | Sfrutta il costruttore di prenotazione per poter aggiungere prenotazioni alla lista associata. |
| **Processing** | Fare dopo |
| **Data** | Ha un attributo IDcampo univoco rispetto alla struttura di appartenenza.  Ha un attributo Sport che specifica lo sport praticabile.  Ha un attributo Prezzo che specifica il costo orario.  Ha un attributo nroMaxPerosne che specifica il numero massimo di persone che possono giocare contemporaneamente.  Possiede la lista ListaPrenotazioneCampi. |

|  |  |
| --- | --- |
| **Identification** | Spogliatoio |
| **Type** | Classe |
| **Purpose & Function** | Rappresenta gli spogliatoi di una struttura e prenotabili dagli utenti.  Permette di aggiungere prenotazioni modificando la lista ListaPrenotazioneSpogliatoi associata.  Permette di stampare le prenotazioni associate. |
| **Dependencies** | Usa la classe Prenotazione.  È usata dalla classe Struttura. |
| **Interfaces** | Sfrutta il costruttore di prenotazione per poter aggiungere prenotazioni alla lista associata. |
| **Processing** | Fare dopo |
| **Data** | Ha un attributo Numero univoco rispetto alla struttura di appartenenza.  Ha un attributo Prezzo che specifica il costo orario.  Possiede una lista ListaPrenotazioneSpogliatoi. |

|  |  |
| --- | --- |
| **Identification** | Registro Strutture |
| **Type** | Classe |
| **Purpose & Function** | Raccoglie tutte le strutture della piattaforma.  Permette di effettuare l’autentificazione di una struttura. |
| **Dependencies** | Usa la classe Struttura.  È usata dalla classe Cliente quando si effettuano prenotazioni. |
| **Interfaces** | Non essendoci la possibilità di aggiungere nuove strutture alla piattaforma non utilizza il costruttore della classe Struttura. |
| **Processing** | Fare dopo |
| **Data** | Ha una lista di strutture chiamata ListaStrutture. |

|  |  |
| --- | --- |
| **Identification** | Cliente |
| **Type** | Classe |
| **Purpose & Function** | Identifica un utente della piattaforma.  Permette di settare i parametri di prenotazione e quindi di effettuare nuove prenotazioni presso le strutture della piattaforma.  Controlla se è stato bannato da una determinata struttura. |
| **Dependencies** | Usa la classe Prenotazione.  È usata dalla classe Struttura. |
| **Interfaces** | Sfrutta il costruttore di prenotazione per poter creare nuove prenotazioni. |
| **Processing** | Fare dopo |
| **Data** | Ha un attributo Nome e uno Cognome che identificano l’utente.  Ha un attributo CodiceFiscale univoco. |

|  |  |
| --- | --- |
| **Identification** | Parametri Prenotazione |
| **Type** | Parametro |
| **Purpose & Function** | Permette al cliente di specificare le preferenze di prenotazione.  Permette al Registro Strutture di controllare la disponibilità rispettando le scelte del cliente. |
| **Dependencies** | È un parametro di una funzione della classe Cliente (controlloDisponibilità).  Usato dalla classe Cliente. |

|  |  |
| --- | --- |
| **Identification** | Codice Fiscale |
| **Type** | Attributo |
| **Purpose & Function** | Identifica univocamente il cliente.  Serve alla classe Struttura per fare il conteggio delle prenotazioni ed eventualmente per effettuare il ban. |
| **Dependencies** | È usato dalla classe Struttura. |

|  |  |
| --- | --- |
| **Identification** | IDstruttura |
| **Type** | Attributo |
| **Purpose & Function** | Identifica univocamente una struttura.  Serve alla classe Registro Strutture per fare l’autentificazione della struttura. |
| **Dependencies** | È usato dalla classe Registro Strutture. |

|  |  |
| --- | --- |
| **Identification** | Prenotazione |
| **Type** | Classe |
| **Purpose & Function** | Serve per poter creare una nuova prenotazione relativa ad un campo ed eventualmente anche ad uno spogliatoio.  Permette di calcolare il costo della prenotazione basandosi sul prezzo orario del campo (e dello spogliatoio) e tenendo presente un eventuale sconto del cliente. |
| **Dependencies** | È usata dalle classi Cliente, Campo, Spogliatoio. |
| **Processing** | Fare dopo |
| **Data** | Ha un attributo univoco CodicePrenotazione.  Ha gli attributi NomeStruttura, Data, Orario, Sport, Spogliatoio che la caratterizzano e rappresentano le preferenze del cliente.  Ha gli attributi NomeCampo, NomeSpogliatoio che vengono assegnati in base alla disponibilità dalla struttura specificata durante la fase di ricerca.  Ha un attributo prezzo da calcolare con il metodo calcolaPrezzo.  Ha un attributo CodiceFiscale dell’utente che ha effettuato la prenotazione. |