



**UNIVERSITÀ
DEGLI STUDI
DI BERGAMO**

Dipartimento di Ingegneria Gestionale, dell'Informazione e della Produzione

Corso di laurea magistrale in Ingegneria Informatica

PROGETTO P.A.C. PALESTRA DIGITALIZZATA

Andrea Appiani
Matricola 1057683

Fabio Filippo Mandalari
Matricola 1047426

A.A. 2021/2022

IDEA DI PROGETTO

Il progetto è stato pensato intorno all'ambiente fitness di una generica palestra. Nella fattispecie, l'obiettivo primario del progetto è quello di digitalizzare le schede di allenamento degli utenti di una generica palestra. Questo scopo porta all'introduzione di un certo livello di interattività delle schede con i macchinari che popolano la palestra.

Affinché le schede digitalizzate possano essere consultabili dagli utenti si è optato per l'implementazione di un client che le possa mostrare in chiaro sullo schermo. Oltre a mostrare la scheda si è voluto implementare un sistema che in ogni momento, su richiesta dell'utente, sia in grado di aggiornare la scheda fitness sostituendo ad ogni macchinario occupato un altro macchinario equivalente che sia libero o che si libererà a breve.

Gestendo informazioni sullo stato di ogni macchinario presente nella palestra si è anche voluto implementare una dashboard che mostri lo stato corrente dei macchinari a tutti i presenti. La dashboard è un display virtuale mostrante un set specifico di informazioni.

TOOLCHAIN

Strumenti di comunicazione usati per comunicazioni e brainstorming:

- Google Meet
- Telegram

Repository online:

- GitHub

Linguaggio e IDE usati per l'implementazione del codice:

- Java
- Eclipse

Framework utilizzati:

- Spring
- Swing

Plugin di Eclipse utilizzati:

- Window Builder
- Maven
- JUnit
- JAutodoc

Software per la creazione di diagrammi in formato UML:

- AstahUML

INDICE

ITERAZIONE 0

ANALISI DEI REQUISITI

USE CASE STORIES

FASE 1: RICONOSCIMENTO DI TUTTI GLI ATTORI

FASE 2: ELENCAZIONE DI TUTTI I POSSIBILI CASI D'USO

FASE 3: DESCRIZIONE DI TUTTI I CASI D'USO

USE CASE DIAGRAM

ANALISI DELL'ARCHITETTURA

TOPOLOGY DIAGRAM IN STILE LIBERO

TOPOLOGY DIAGRAM FORMALE

SCELTE DI IMPLEMENTAZIONE

ITERAZIONE 1

INTRODUZIONE

RAGGRUPPAMENTO CASI D'USO

COMPONENT DIAGRAM

DIAGRAMMA DELLE INTERFACCE

DATA CLASS DIAGRAM

STILE ARCHITETTURALE

VARIAZIONE RISPETTO ALL'IDEA ORIGINALE

ITERAZIONE 2

INTRODUZIONE

IMPLEMENTAZIONE

DESCRIZIONE DEI PASSI

ITERAZIONE 3

INTRODUZIONE

ASSUNZIONI FATTE

IMPLEMENTAZIONE

CASI DI TEST

COPERTURA DEI TEST

ITERAZIONE 4

ITERAZIONE 5

INTRODUZIONE

IMPLEMENTAZIONE

IMPLEMENTAZIONE ALGORITMO

PSEUDOCODICE

COMPLESSITÀ ALGORITMO

CASI DI TEST

COPERTURA DEI TEST

GUIDA ALL'INSTALLAZIONE

ITERAZIONE 0

ANALISI DEI REQUISITI

Per effettuare l'analisi dei requisiti è stato necessario immedesimarsi nei panni di un utente per poter prendere in considerazione tutti i possibili scenari e tutti i requisiti congeniali alla realizzazione del progetto.

Per fare ciò siamo ricorsi all'uso delle *use case stories*, ovvero una breve spiegazione di ogni funzionalità dell'applicazione software dal punto di vista di ciascuno degli attori che hanno a che fare con tale funzionalità.

USE CASE STORIES

L'elaborazione delle use case stories ha richiesto tre fasi distinte:

1. Riconoscimento di tutti gli attori;
2. Elencazione di tutti i possibili casi d'uso;
3. Descrizione di tutti i casi d'uso.

FASE 1: RICONOSCIMENTO DI TUTTI GLI ATTORI

Gli attori coinvolti nel sistema, suddivisi tra “primario” e “secondario” in base all'impatto che il particolare attore ha all'interno del progetto, sono:

- Utente (primario);
- Trainer (secondario);
- Amministratore di sistema (secondario);
- Sistema di autenticazione (secondario).

FASE 2: ELENCAZIONE DI TUTTI I POSSIBILI CASI D'USO

I casi d'uso sono stati concepiti immedesimandosi nei panni di tutti gli attori individuati nella fase 1:

- UC1: Entrata in palestra;
- UC2: Registrazione;
- UC3: Login;
- UC4: Logout;
- UC5: Gestione account;
- UC6: Generazione scheda;
- UC7: Visualizzazione scheda;
- UC8: Calcolo della schedulazione ottima;
- UC9: Visualizzazione schedulazione;
- UC10: Visualizzazione stato macchinari;
- UC11: Aggiorna stato uso macchinario;
- UC12: Overview utenti registrati;
- UC13: Generazione report.

FASE 3: DESCRIZIONE DI TUTTI I CASI D'USO

UC1: ENTRATA IN PALESTRA

Attori coinvolti: utente, trainer.

POV utente: come utente voglio poter entrare in palestra.

POV trainer: come trainer voglio poter entrare in palestra.

UC2: REGISTRAZIONE

Attori coinvolti: utente, trainer, sistema di autenticazione (AS, Authentication Server).

POV utente: come utente voglio poter creare il mio account personale.

POV trainer: come trainer voglio poter creare il mio account personale.

POV AS: come sistema di autenticazione devo poter permettere a qualunque cliente della palestra di creare un account personale.

UC3: LOGIN

Attori coinvolti: utente, trainer, sistema di autenticazione (AS, Authentication Server).

POV utente: come utente voglio poter effettuare il login al mio account sull'applicazione.

POV trainer: come trainer voglio poter effettuare il login al mio account sull'applicazione.

POV AS: come sistema di autenticazione devo poter permettere agli utenti registrati di effettuare il login al proprio account.

UC4: LOGOUT

Attori coinvolti: utente, trainer, sistema di autenticazione (AS, Authentication Server).

POV utente: come utente voglio poter effettuare il logout dal mio account sull'applicazione.

POV trainer: come trainer voglio poter effettuare il logout dal mio account sull'applicazione.

POV AS: come sistema di autenticazione devo poter permettere agli utenti registrati di effettuare il logout dal proprio account.

UC5: GESTIONE ACCOUNT

Attori coinvolti: utente, trainer.

POV utente: come utente voglio poter modificare le informazioni presenti sul mio profilo.

POV trainer: come trainer voglio poter modificare le informazioni presenti sul mio profilo.

UC6: GENERAZIONE SCHEDA

Attori coinvolti: utente, trainer.

POV utente: come utente voglio poter richiedere la generazione di una nuova scheda.

POV trainer: come trainer voglio poter richiedere la generazione di una nuova scheda.

UC7: VISUALIZZAZIONE SCHEDA

Attori coinvolti: utente, trainer.

POV utente: come utente voglio poter visualizzare la mia scheda di allenamento personale.

POV trainer: come trainer voglio poter visualizzare le schede di allenamento degli utenti.

UC8: CALCOLO DELLA SCHEDULAZIONE OTTIMA

Attori coinvolti: utente.

POV utente: come utente voglio poter richiedere una nuova schedulazione se uno dei macchinari presenti nella mia scheda è al momento occupato.

UC9: VISUALIZZAZIONE SCHEDULAZIONE

Attori coinvolti: utente.

POV utente: come utente voglio poter visualizzare la schedulazione aggiornata.

UC10: VISUALIZZAZIONE STATO MACCHINARI

Attori coinvolti: utente, trainer, amministratore di sistema (SA, System Administrator).

POV utente: come utente voglio poter visualizzare lo stato di occupazione dei macchinari in qualunque momento tramite una dashboard.

POV trainer: come trainer voglio poter visualizzare lo stato di occupazione dei macchinari in qualunque momento tramite una dashboard.

POV SA: come amministratore di sistema voglio poter visualizzare lo stato di occupazione dei macchinari in qualunque momento.

UC11: AGGIORNA STATO USO MACCHINARIO

Attori coinvolti: utente.

POV utente: come utente voglio poter trasmettere quale sia lo stato di ogni macchinario premendo uno dei tre tasti presenti su ognuno di essi (occupato, libero, guasto).

UC12: OVERVIEW UTENTI REGISTRATI

Attori coinvolti: amministratore di sistema (SA, System Administrator).

POV SA: come amministratore di sistema voglio avere accesso al database contenente le informazioni degli utenti registrati.

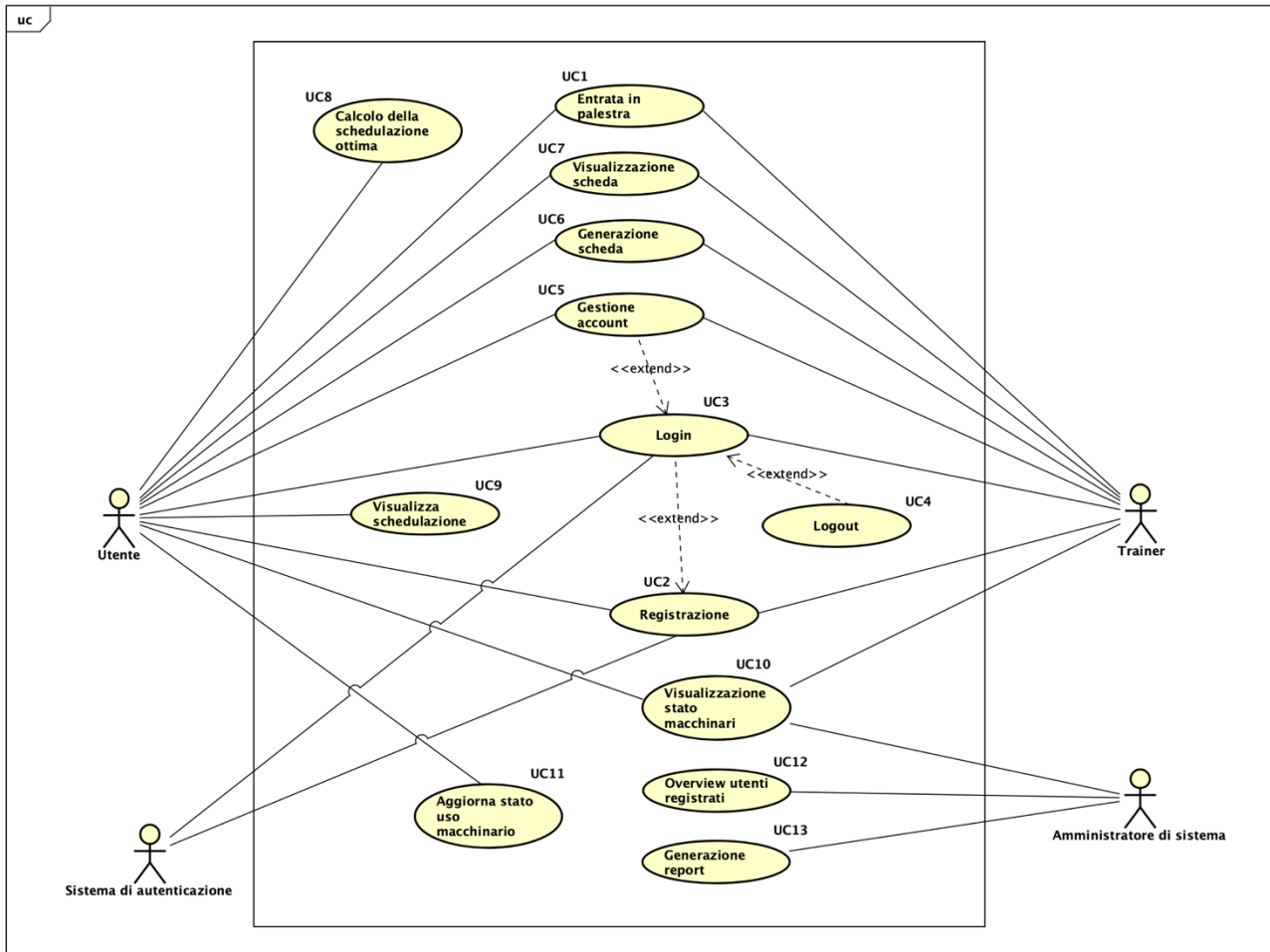
UC13: GENERAZIONE REPORT

Attori coinvolti: amministratore di sistema (SA, System Administrator).

POV SA: come amministratore di sistema voglio poter generare dei report relativi alla profilazione della palestra.

USE CASE DIAGRAM

Partendo dalle use case stories è stato redatto lo use case diagram, ovvero un diagramma in cui vengono specificati tutti i requisiti funzionali che il sistema dovrà implementare:



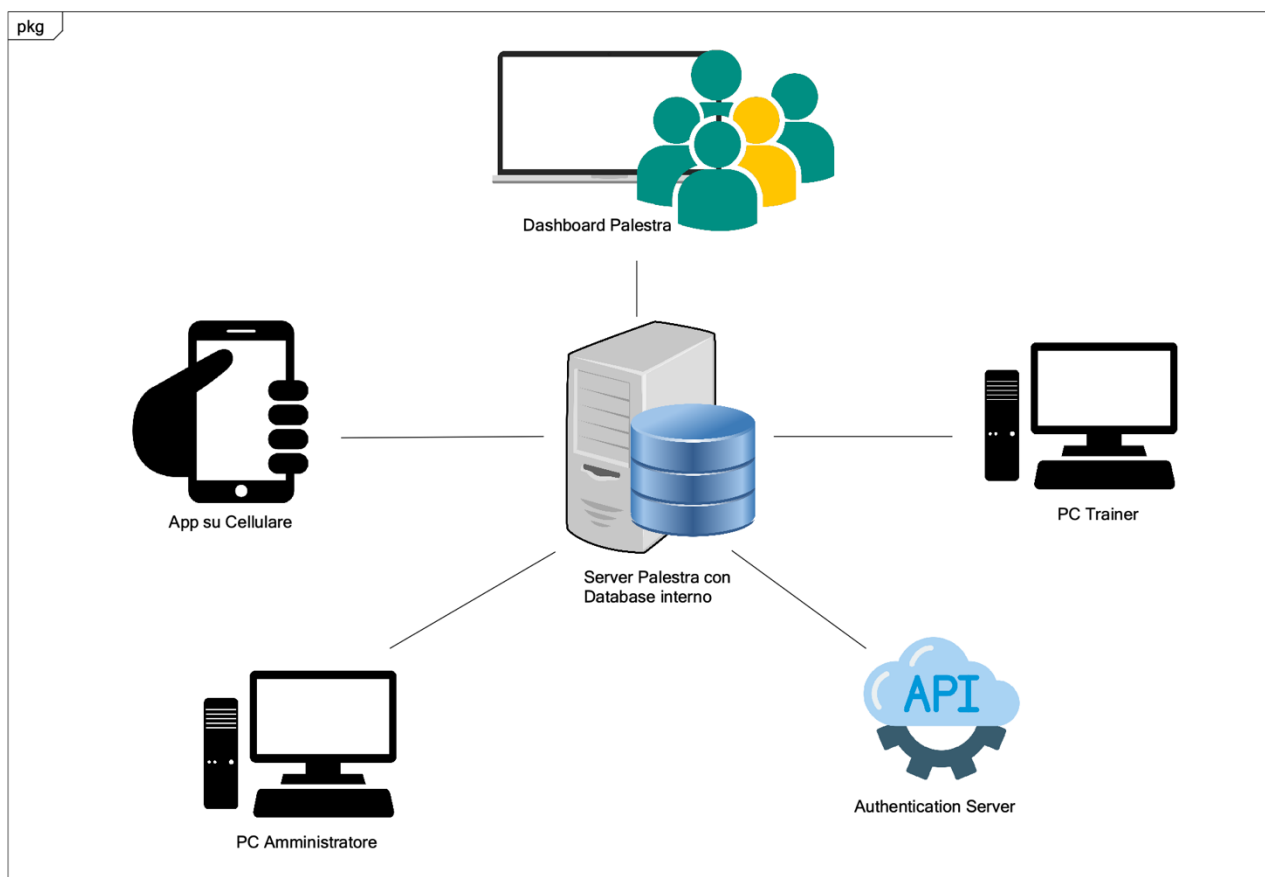
ANALISI DELL'ARCHITETTURA

Per quanto riguarda l'architettura sono stati realizzati due topology diagram:

- il primo, meno formale, è stato utile al team per creare una idea generale dell'intero sistema;
- il secondo, più formale, è servito per mostrare nel dettaglio l'allocazione delle componenti software su quelle hardware.

TOPOLOGY DIAGRAM IN STILE LIBERO

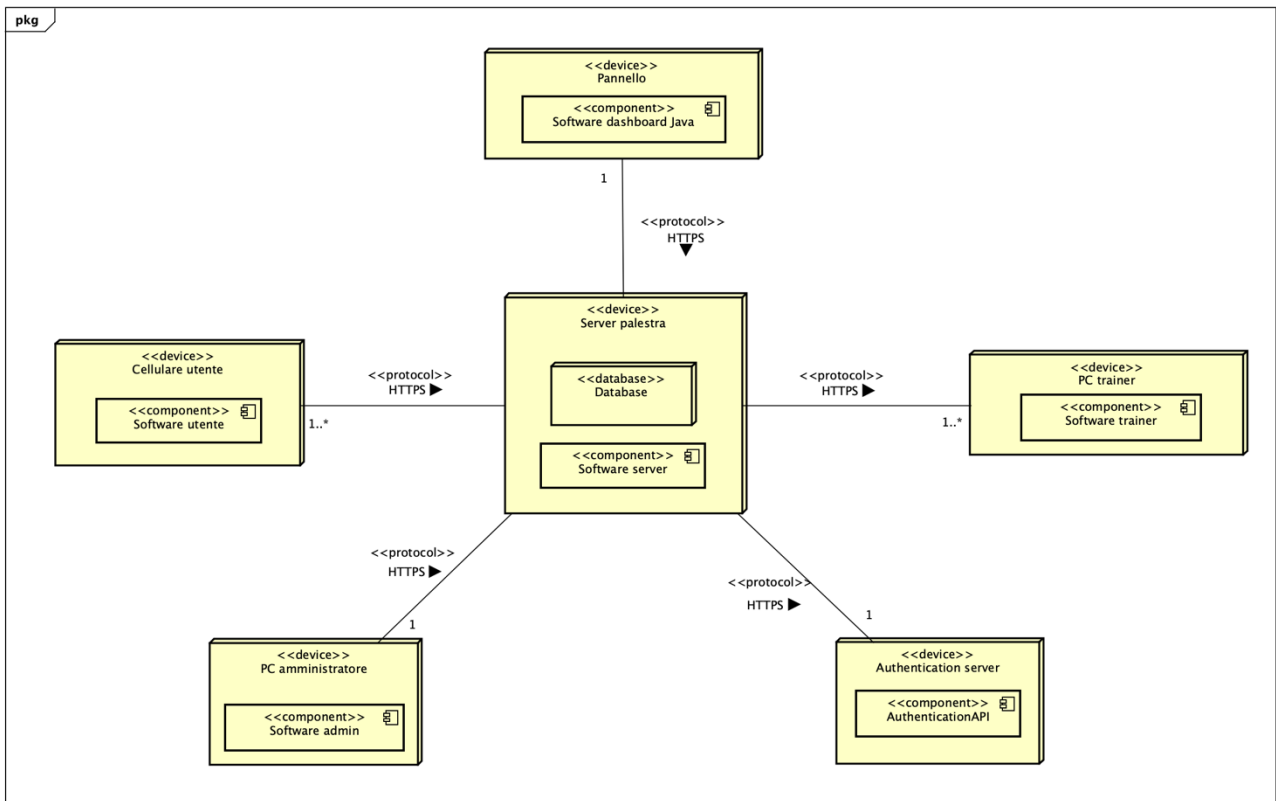
Questa prima rappresentazione, del tutto priva di formalismo, si pone come solo obiettivo quello di mettere in evidenza i dispositivi considerati nell'architettura, permettendo così di fornire una visualizzazione ad alto livello delle componenti del sistema e della loro interazione.



TOPOLOGY DIAGRAM FORMALE

Questo topology diagram si propone di rappresentare le medesime informazioni messe in evidenza nel diagramma precedente, ma esplicitando altri due ulteriori aspetti:

- cosa è da considerare come “device” e cosa come “software component”;
- la tecnologia comunicativa utilizzata tra i diversi “device”.



SCELTE DI IMPLEMENTAZIONE

Nel proseguo del progetto si è scelto di porre particolare attenzione a determinati casi d'uso:

<u>USE CASE</u>	<u>IMPLEMENTAZIONE</u>
UC1	NO
UC2	SI
UC3	SI
UC4	SI
UC5	NO
UC6	NO
UC7	SI
UC8	SI
UC9	SI
UC10	SI
UC11	SI
UC12	NO
UC13	NO

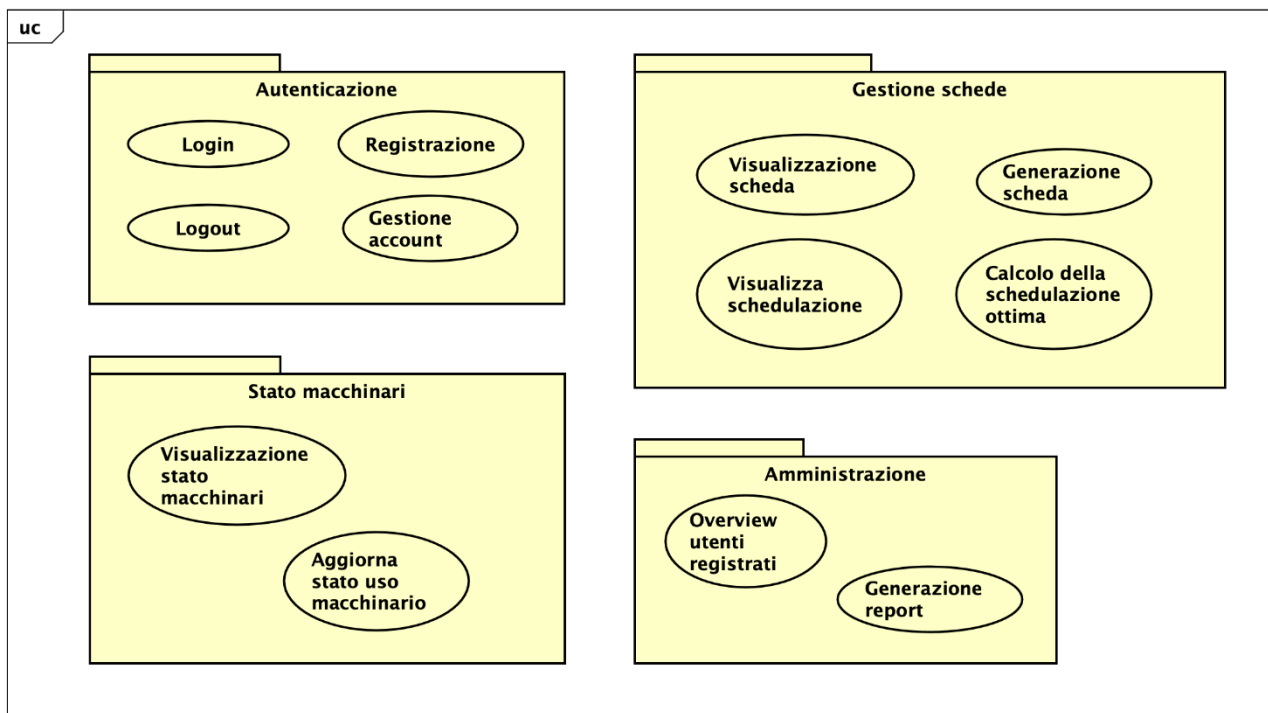
ITERAZIONE 1

INTRODUZIONE

Nell'ambito della seguente iterazione abbiamo cercato di dare una forma più concreta all'intera architettura guardando al sistema nella sua totalità.

RAGGRUPPAMENTO CASI D'USO

Il punto di partenza è stato un raggruppamento dei casi d'uso in gruppi che ne contenessero una pluralità sulla base di un ben determinato criterio. Il risultato è il seguente:



COMPONENT DIAGRAM

Il diagramma delle componenti si pone come obiettivo quello di porre l'attenzione su come i componenti del sistema interagiscono tra di loro. Si sono innanzitutto divisi i componenti di ogni sotto-sistema in tre livelli: interfaccia grafica, business logic e data layer; successivamente tramite la notazione “ball and socket” è stato messo in risalto quale componente espone una data interfaccia e quale componente invece ne usufruisce:

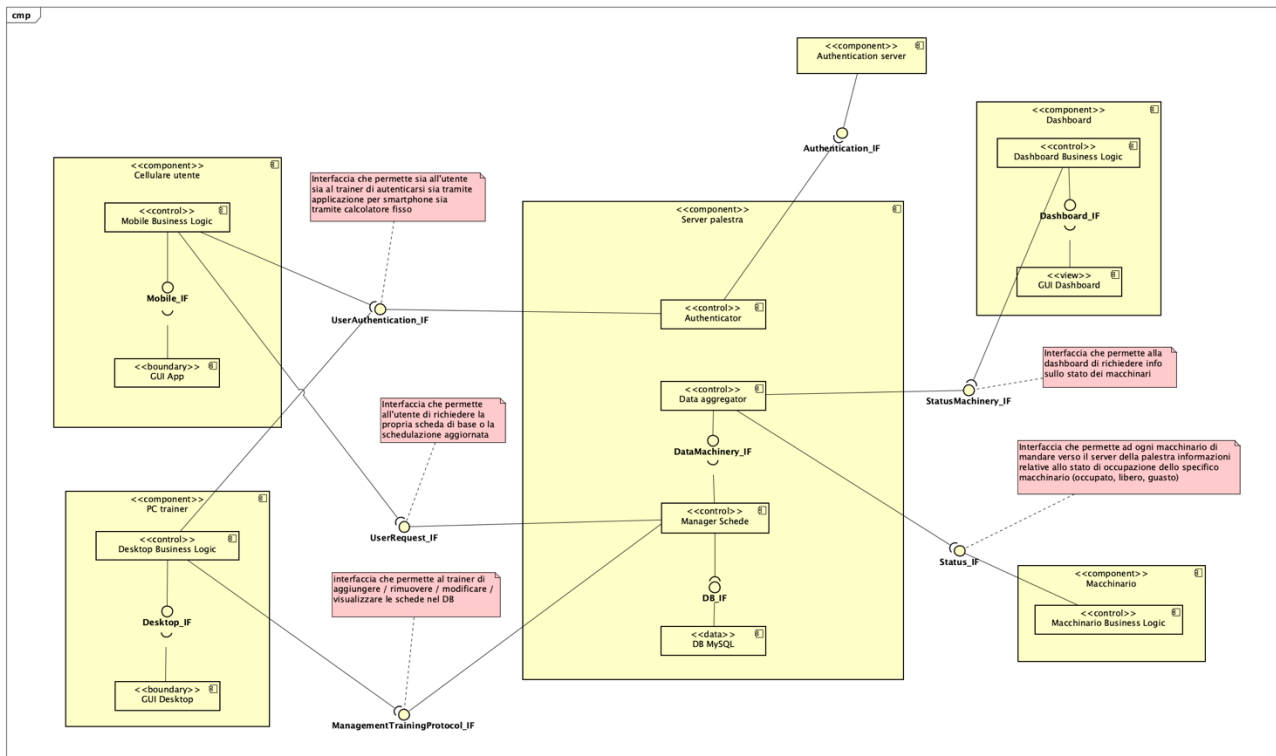
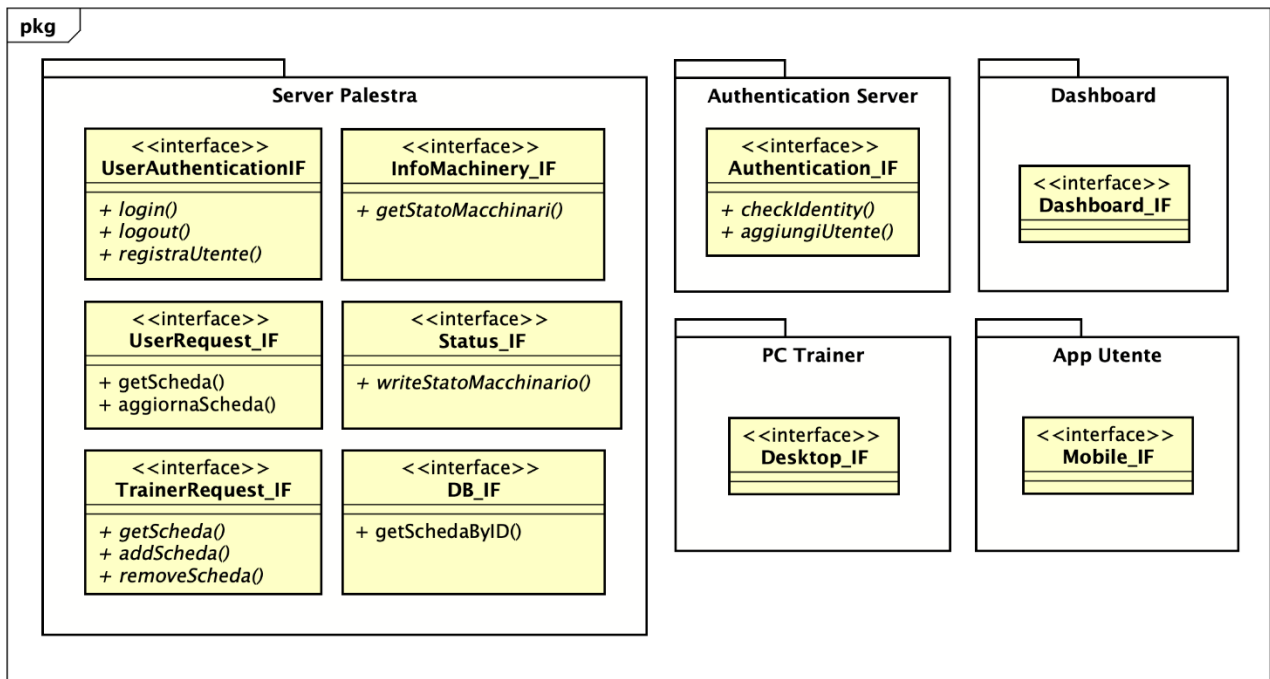


DIAGRAMMA DELLE INTERFACCE

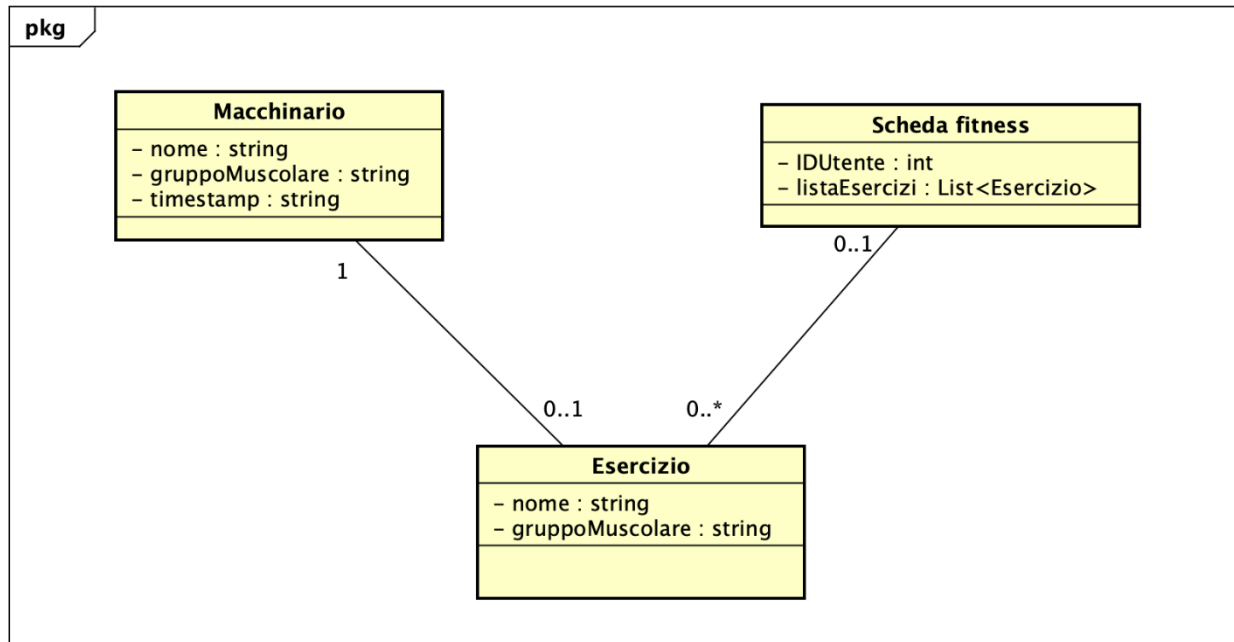
Con il diagramma delle interfacce si è voluto definire le interfacce di cui ogni componente del diagramma presentato nel paragrafo precedente si serve per effettuare tutte le operazioni necessarie al suo funzionamento. Ogni interfaccia prevede al suo interno una serie di metodi che, allo stato attuale, sono rappresentati con una segnatura semplificata. La loro implementazione definitiva verrà definita solo durante le successive fasi di sviluppo del processo agile.



DATA CLASS DIAGRAM

Il diagramma delle classi consente di descrivere le principali componenti del sistema permettendo di mettere in evidenza caratteristiche ed eventuali relazioni fra loro.

Le classi caratteristiche del nostro progetto sono:



STILE ARCHITETTURALE

In prima battuta si è deciso di adottare come stile architetturale per la creazione di tutte le componenti che costituiscono progetto il pattern MVC (Model View Controller). Per questo motivo è stato scelto di utilizzare il **singleton** per tutte quelle classi che svolgono i ruoli di Model, View e Controller, in quanto uniche all'interno del sistema in cui si trovano.

Un altro stile architetturale adottato per questo progetto è il "Multi-Layer" il quale, come si può notare nel diagramma dei componenti, comporta in ogni sotto-sistema la suddivisione verticale nei livelli:

- User Interface (View)
- Application Logic (Controller)
- Database Access (Model)

VARIAZIONE RISPETTO ALL'IDEA ORIGINALE

Il progetto è stato originariamente pensato intorno all'uso di una applicazione Android per il client, ma durante l'implementazione è stato scelto di continuare con un generico terminale Java.

ITERAZIONE 2

INTRODUZIONE

L'iterazione 2 è stata dedicata allo sviluppo e all'implementazione dei casi d'uso relativi all'autenticazione dell'utente e del trainer (UC2, UC3, UC4).

IMPLEMENTAZIONE

Il punto di partenza è stata la creazione, in Java, di un client MVC-based rappresentante l'app che è in esecuzione sul terminale dell'utente. Il client in questione consta di una View mostrante i pulsanti di login, logout e registrazione (con i relativi campi testuali per l'inserimento) e di un Controller in ascolto su questi pulsanti.

Secondo MVC il Controller dispone di diversi action listeners in ascolto sui pulsanti della View, incaricati di inviare la rispettiva richiesta REST tramite URL.

Nel caso della richiesta di login, l'URL viene generato nel seguente modo:

```
http://localhost::8080/autentica? + view.getNome() + "&" +  
view.getPassword()
```

Dal punto di vista del server abbiamo creato un REST Controller denominato Controller Autenticazione (componente mock). Esso consta di un'interfaccia avente i seguenti metodi mappati a determinati URL:

- `login(String nome, String password);`
- `logout(String nome, String password);`
- `registraUtente(String nome, String password).`

DESCRIZIONE DEI PASSI

Una volta avviata l'app compare un'interfaccia in cui l'utente, sia esso un cliente della palestra o un trainer, deve inserire i propri dati per poter eseguire l'accesso tramite il pulsante di Login oppure per registrarsi tramite il pulsante di Registrazione. Premuto uno di questi pulsanti, viene inviata una richiesta tramite URL al corrispondente metodo offerto dall'API del server sopra mostrata.

Il Controller Autenticazione, ricevendo queste richieste, delega l'autenticazione al server LDAP tramite l'API da questo esposta. L'esito viene ritornato al Controller Autenticazione il quale, a sua volta, lo restituisce al Controller dell'applicazione utente:

- in caso di esito positivo la View passa ad una schermata contenente le informazioni sulla scheda di training dell'utente;
- in caso di esito negativo la View mostra un messaggio di errore.

Infine, sulla schermata della scheda di training è presente un pulsante di Logout tramite il quale, similmente agli altri due, viene inviata una richiesta al Controller Autenticazione per terminare la sessione dell'utente.

ITERAZIONE 3

INTRODUZIONE

L'iterazione 3 è dedicata allo sviluppo e all'implementazione dei casi d'uso relativi alla visualizzazione dello stato dei macchinari (UC10) e all'aggiornamento del loro stato di uso (UC11).

ASSUNZIONI FATTE

Per poter conoscere lo stato attuale di un macchinario si è deciso di assumere innanzitutto che su ognuno di essi siano presenti i tre pulsanti “Libera Macchinario”, “Occupa Macchinario” e “Macchinario Guasto” che comunicano al server lo stato corrente quando premuti da un utente.

In vista del caso d'uso legato alla generazione di una scheda fitness aggiornata in base allo stato corrente dei macchinari (UC8), si è scelto di inviare le informazioni nel seguente formato:

- nome del macchinario \rightarrow String
- tipologia del macchinario \rightarrow String
- orario al quale si libera il macchinario \rightarrow String

L'informazione legata all'orario assume uno specifico valore in una delle seguenti situazioni:

- Premendo “LIBERA MACCHINARIO” viene inviato al server l'orario attuale;
- Premendo “OCCUPA MACCHINARIO” viene inviato al server un orario ottenuto effettuando internamente il calcolo “orario attuale + X minuti”, in cui la stima di X potrebbe essere fatta osservando l'occupazione media di quel macchinario;
- Premendo “MACCHINARIO GUASTO” viene inviato al server l'orario 23:59:59, assicurando la minore priorità possibile.

In questo modo è possibile gestire un sistema di priorità tra i macchinari utilizzando l'orario dal quale sono/saranno liberi.

IMPLEMENTAZIONE

Per permettere agli utenti e al trainer di visualizzare lo stato corrente dei macchinari è stato creato un client Dashboard che si occupa di mostrare a schermo in formato tabellare le informazioni sui macchinari aggiornandole ogni 5 secondi.

In fase di implementazione della Dashboard si è usato il paradigma MVC per creare sia una View sia un Controller. Il Controller si occupa di inviare periodicamente richieste all'URL `http://localhost:8080/getStatoMacchinari` e di processare la stringa JSON ricevuta in risposta attraverso il metodo di parsing `parsingMacchinari(String s)`.

La classe `Macchinario` è stata implementata in modo tale da possedere un metodo `getStato()` che possa restituire la stringa "OCCUPATO", "LIBERO" o "GUASTO" confrontando il timestamp del macchinario con l'orario attuale.

Un possibile output prodotto dalla View della Dashboard è il seguente:

Stato dei macchinari:		
Macchinario	Gruppo Muscolare	Stato
Panca piana	Pettorali	LIBERO
Colonna	Braccia	LIBERO
Cavi	Braccia	LIBERO
Leg-press	Gambe	OCCUPATO
Leg-extension	Gambe	LIBERO
Panca inclinata	Pettorali	GUASTO

Server-side invece è stato creato il componente "Controller Aggregatore" come REST Controller che contiene un `ArrayList` di macchinari che aggiorna ogni volta che riceve nuove informazioni da un macchinario. Inoltre, Controller Aggregatore mette a disposizione due API:

- `InfoMachineryIF` permette alla Dashboard di leggere la lista di macchinari aggiornata:

```
@GetMapping("/getStatoMacchinari")
RisorsaJSON getStatoMacchinari()
```

- `StatusIF` permette ad ogni macchinario di aggiornare il proprio stato sulla lista:

```
@GetMapping("/writeStatoMacchinario")
writeStatoMacchinario()
```









CASI DI TEST

Il componente Controller Aggregatore è stato testato tramite unit-testing sul metodo `writeStatoMacchinario()` dando in input diversi macchinari, eseguendo quindi il metodo più volte e controllando poi che la lista sul Controller venga correttamente aggiornata.















Il componente Controller Dashboard è stato invece testato sul metodo `parsingMacchinari()` (metodo che si occupa di ritornare una lista di macchinari partendo dalla stringa JSON ricevuta dal Controller Aggregatore) dando ad esso in input input una stringa JSON rappresentante diversi macchinari, controllando che la lista ritornata sia rappresentata correttamente nella Dashboard View.

COPERTURA DEI TEST

Copertura del test effettuato sul Controller Dashboard:

Element	Coverage	Covered Instructions	Missed Instructions	Total Instructions
▼ ProgettoPAC-Dashboard	 61,3 %	298	188	486
▼ src/main/java	 58,1 %	254	183	437
▼ com.example.demo	 58,1 %	254	183	437
> DashboardController.java	 41,5 %	59	83	142
> DashboardView.java	 78,9 %	146	39	185
> Macchinario.java	 57,0 %	49	37	86
> DashboardApplication.java	 0,0 %	0	24	24
> src/test/java	 89,8 %	44	5	49

Copertura del test effettuato sul Controller Aggregatore:

Element	Coverage	Covered Instructions	Missed Instructions	Total Instructions
▼ ProgettoPAC-ServerJSON	 23,9 %	299	950	1.249
▼ src/main/java	 23,9 %	245	778	1.023
▼ com.example.demo	 23,9 %	245	778	1.023
> ControllerManagerSchede.java	 0,0 %	0	426	426
> JSONParser.java	 2,2 %	3	132	135
> SchedaFitness.java	 0,0 %	0	88	88
> Esercizio.java	 0,0 %	0	46	46
> Macchinario.java	 63,7 %	51	29	80
> Model.java	 0,0 %	0	18	18
> RisorsaJSON.java	 0,0 %	0	16	16
> ControllerAutenticazione.java	 0,0 %	0	8	8
> ServerApplication.java	 0,0 %	0	8	8
> ControllerAggregatore.java	 96,5 %	191	7	198
> src/test/java	 23,9 %	54	172	226

ITERAZIONE 4

Questa iterazione è stata dedicata all'implementazione del caso d'uso UC7 relativo alla richiesta di visualizzazione della propria scheda da parte di un utente.

Questo ha portato ad aggiungere: una nuova schermata sulla view lato client per visualizzare le schede, un componente ControllerManagerSchede (controller REST) lato server e un model (componente mock) sempre lato server che insieme al controller si occupa di ritornare dal database le schede richieste dall'utente.

ITERAZIONE 5

INTRODUZIONE

L'iterazione 5 è stata dedicata allo sviluppo e all'implementazione dei casi d'uso relativi alla richiesta di aggiornamento della scheda da parte dell'utente sulla base dello stato attuale dei macchinari. I casi d'uso in questione sono UC8 e UC9.

IMPLEMENTAZIONE

Dal punto di vista dell'app dell'utente è stata aggiornata la View aggiungendo i pulsanti “Scheda Originale” e “Scheda Aggiornata” per permettere all'utente di passare tra una versione e l'altra della propria scheda fitness. I campi testuali dove vengono visualizzati gli esercizi della scheda aggiornata sono gli stessi di quelli usati per la scheda originale ed è per questo che non sono richieste ulteriori modifiche alla View.

Sul Controller del client sono stati aggiunti i rispettivi Action Listener per inviare le richieste tramite URL al componente lato server denominato ManagerSchede (REST Controller).

Il Manager Schede è già stato implementato nell'iterazione precedente per supportare i casi d'uso relativi alla richiesta di schede fitness originali da ritornare all'utente, perciò, nel contesto dell'iterazione corrente, è sufficiente implementare l'API REST per rispondere alle richieste di aggiornamento delle schede:

```
@GetMapping("/aggiornaScheda")
public RisorsaJSON aggiornaScheda(int id)
```

Il metodo aggiornaScheda() ricevendo l'id dell'utente e richiede sia la rispettiva scheda fitness al DB sia la lista dei macchinari con lo stato attuale per produrre la nuova lista di esercizi. RisorsaJSON è un'istanza contenente sia l'ID della richiesta sia la stringa JSON rappresentante la lista di esercizi.

Esempio di RisorsaJSON (in formato JSON) inviata con id = 1 e un solo esercizio nella lista:

```
{ "lista": "[{\\"tipo\\":\\"Braccia\\",\\"nome\\":\\"Flessioni\\"}]", "id":1
```

Essendo il fetching della scheda fitness dal DB un'operazione onerosa (considerato che la richiesta di aggiornamento scheda può essere effettuata più volte in qualsiasi momento) è stata creata una lista dinamica di SchedaFitness all'interno del controller Manager Schede. In questo modo una scheda fitness viene estratta dal DB solo la prima volta per poi essere salvata in questa lista permettendo un accesso molto più veloce nel caso debba essere aggiornata in seguito.

La lista di macchinari con lo stato aggiornato viene ottenuta tramite il Controller Aggregatore usando la stessa API usata dalla Dashboard, ovvero InfoMachineryIF.

IMPLEMENTAZIONE ALGORITMO

Una volta che la richiesta viene ricevuta dal metodo `aggiornaScheda()` l'effettiva creazione della lista di esercizi viene affidata ad un metodo implementato secondo la metodologia Greedy:

```
algoritmoGreedy(SchedaFitness S, List<Macchinario> Lm)
```

L'output è una lista di macchinari che viene convertita in esercizi per l'invio al client dell'utente.

PSEUDOCODICE ALGORITMO

```
algoritmoGreedy(EserciziScheda S [1..n], Lista macchinari [1..m])
    Lista aggiornata // Contiene i macchinari selezionati dall'algoritmo
    Coda 1 ... K // È creata una PriorityQueue per ogni categoria "k" di macchinari
    foreach(macchinario m[k])
        C[k].enqueue(m);
    foreach(Esercizio E[k] in S)
        if (M[k] libero)
            aggiornata.add(M);
        else
            Macchinario M ← Coda[k].dequeue();
            if(M != null) aggiornata.add(M);
    endif
    return aggiornata;
```

M[k] è il macchinario corrispondente a quell'esercizio: se questo è libero non viene sostituito con un altro a prescindere dalla priorità.

COMPLESSITÀ ALGORITMO

Usando la PriorityQueue del JCF le operazioni di `enqueue()` e `dequeue()` risultano avere costo $O(\log(m))$, dove in questo caso "m" è il numero di macchinari contenuti in esse; usando invece una LinkedList per contenere la lista risultante "aggiornata" permette di avere costo unitario $O(1)$ per l'operazione di inserimento `add()`.

La prima sezione dell'algoritmo, con la creazione e riempimento delle code, ha quindi costo $m * \log(m)$ immaginando come caso peggiore quello per cui tutte le code devono essere riempite con tutti i macchinari.

La seconda sezione, invece, partendo con l'iterazione sugli "n" esercizi della scheda utente, esegue "n" volte l'operazione di `dequeue()` e di `add()` per un costo totale di $n * (\log(m) + 1)$. Questo considerando ancora il caso peggiore, ovvero il caso per cui viene sempre eseguito il ramo "else".

Complessivamente, il costo dell'algoritmo è: $T(n, m) = m * \log(m) + n * (\log(m) + 1)$.

Siccome si immagina di avere un numero di macchinari "m" maggiore rispetto al numero "n" di esercizi nella scheda, si può approssimare asintoticamente a: $T(m) = O(m * \log(m))$.

CASI DI TEST

I nuovi metodi implementati lato server che richiedono unit-testing sono:

- L'algoritmo greedy;
- Il metodo `macchinarioLibero()` per controllare che un certo macchinario sia libero;
- Il metodo `searchScheda()` per ritornare una scheda dalla lista dinamica memorizzata sul controller "Manager Schede".

TEST algoritmo greedy & `macchinarioLibero()`















Dati in input una scheda con determinati esercizi e una lista di macchinari con determinati orari ai quali si libereranno, si verifica che l'algoritmo dia in output una lista di macchinari che:

1. Abbia sostituito correttamente per ogni esercizio un macchinario della stessa categoria, mantenendo l'ordine della scheda originale;
2. Per ogni categoria sia stato assegnato quello a priorità maggiore (ovvero quello con il timestamp minore);
3. Nel caso l'esercizio avesse il macchinario corrispondente libero al momento dell'esecuzione, questo non viene stato sostituito con un altro macchinario equivalente, questo test include quindi quello per `macchinarioLibero()`.

TEST `searchScheda()`

Sono state aggiunte alla lista di schede del Controller "Manager Schede" diverse schede con i rispettivi ID. Dando in input al metodo `searchScheda(int id)` un id in particolare deve essere ritornata in output la scheda corretta quando questa esiste nella lista, altrimenti "null".

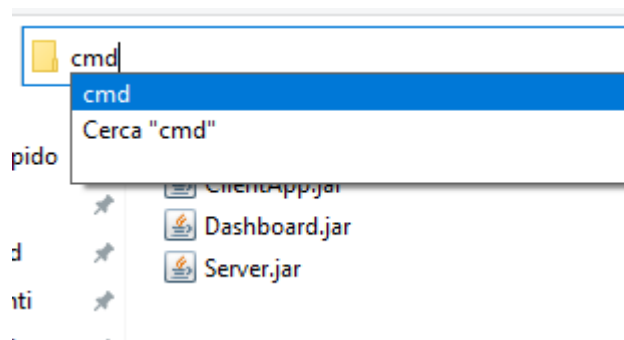
COPERTURA DEI TEST

Element	Coverage	Covered Instructions	Missed Instructions	Total Instructions
ProgettoPAC-ServerJSON	 58,8 %	715	500	1.215
src/main/java	 56,6 %	579	444	1.023
com.example.demo	 56,6 %	579	444	1.023
ControllerManagerSchede.java	 60,8 %	259	167	426
JSONParser.java	 2,2 %	3	132	135
ControllerAggregatore.java	 77,3 %	153	45	198
Macchinario.java	 52,5 %	42	38	80
RisorsaJSON.java	 0,0 %	0	16	16
Esercizio.java	 73,9 %	34	12	46
SchedaFitness.java	 87,5 %	77	11	88
ControllerAutenticazione.java	 0,0 %	0	8	8
ServerApplication.java	 0,0 %	0	8	8
Model.java	 61,1 %	11	7	18
src/test/java	 70,8 %	136	56	192

GUIDA ALL'INSTALLAZIONE

Dati i tre file JAR nella cartella “Codice” si deve scaricarli sul computer preferibilmente all'interno della stessa cartella.

Il primo da avviare per il corretto funzionamento dell'applicazione è Server.jar il quale, per mancanza di interfaccia grafica, è consigliabile avviarlo tramite command window digitando “cmd” nel percorso mostrato nella finestra “esplora risorse”:



Eseguendo poi al suo interno il comando: `java -jar Server.jar`

In questo modo la sua esecuzione verrà facilmente terminata alla chiusura della command window. Successivamente è possibile avviare gli altri due file (anche contemporaneamente) facendo doppio click su di essi.

Questi apriranno un'interfaccia grafica la cui chiusura terminerà l'esecuzione del relativo processo.