

**Libera Università Maria Ss. Assunta**

**Dipartimento di Giurisprudenza, Economia,**

**Politica e Lingue moderne**

**Corso di Laurea Magistrale in ‘Tecniche informatiche per la gestione dei dati’**

**Classe L31**

**Cattedra di**

“Coding Avanzato”

***Titolo Tesi***

ANALISI DEI REQUISITI E DELLA FATTIBILITÀ DI UNA WEB APP PER LA COMPLETA GESTIONE DI SOCIETÀ PODISTICHE AMATORIALI ​

**INGLESE**

**Relatore**

Prof. Massimo Regoli

**Candidato**

Valerio Taddei

Matricola ……………

**Anno Accademico 2021-2022**

Sommario

[***Titolo Tesi*** 1](#_Toc119849409)

[ANALISI DEI REQUISITI E DELLA FATTIBILITÀ DI UNA WEB APP PER LA COMPLETA GESTIONE DI SOCIETÀ PODISTICHE AMATORIALI ​ 1](#_Toc119849410)

[1. Introduzione 5](#_Toc119849411)

[2. Tecnologie e metodologie impiegate 6](#_Toc119849412)

[2.1 Cos’è PHP 7](#_Toc119849413)

[2.2 I metodi GET e POST 8](#_Toc119849414)

[2.3 Symfony Framework 9](#_Toc119849415)

[2.3.1 Pattern MVC 10](#_Toc119849416)

[2.3.2 Dependency injection 12](#_Toc119849417)

[2.4 Postgres 13](#_Toc119849418)

[2.5 Doctrine ORM 14](#_Toc119849419)

[2.6 Bootstrap 15](#_Toc119849420)

[2.8 Docker 16](#_Toc119849421)

[2.7 GitHub 17](#_Toc119849422)

[3. Requisiti del gestionale 19](#_Toc119849423)

[3.1 Requisiti Funzionali 20](#_Toc119849424)

[3.2 UML 21](#_Toc119849425)

[3.2.1 Sequence Diagram 22](#_Toc119849426)

[3.2.2 Use Case Diagram 23](#_Toc119849427)

[Flusso del Front-end 24](#_Toc119849428)

[4. Base dati 25](#_Toc119849429)

[4.1 Schema E-R 26](#_Toc119849430)

[4.2 Schema Logico 28](#_Toc119849431)

[4.3 Creazione del database in locale 32](#_Toc119849432)

[5. Sviluppo della web app 34](#_Toc119849433)

[5.1 Creazione delle tabelle 35](#_Toc119849434)

[5.2 Creazione delle classi PHP 37](#_Toc119849435)

[5.3 Creazione dei repository 38](#_Toc119849436)

[5.4 I primi Controller 39](#_Toc119849437)

[5.4.1 Athletes Controller 40](#_Toc119849438)

[5.4.2 Personal Data Controller 41](#_Toc119849439)

[5.4.3 Security Controller 42](#_Toc119849440)

[5.5 Front-end 43](#_Toc119849441)

[5.5.2 La grafica dell’app e la navigazione 44](#_Toc119849442)

[6. To do – Future implementazioni 45](#_Toc119849443)

[7. Test 47](#_Toc119849444)

[8. Deploy 48](#_Toc119849445)

[9. Mobile App 49](#_Toc119849446)

[10. Bibliografia e sitografia : 50](#_Toc119849447)

[Ringraziamenti 52](#_Toc119849448)

# Introduzione

L’incremento del movimento dei runners inizia intorno alla fine degli anni ’60.

Nel dopoguerra, infatti, la popolazione iniziò a sposare stili di vita finalizzati al benessere fisico e, pertanto, più orientati alla salute; da qui il consolidamento della corsa individuale, più accessibile in assoluto rispetto alle altre attività aerobiche, si diffuse con gradualità in tutte le fasce d’età fino a diventare un fenomeno di massa in tutto il mondo.

Le uniche attività di corsa organizzate avevano lo scopo di distinguere i migliori podisti che avrebbero partecipato ai Giochi Olimpici.

Dagli anni ’70 in poi l’escalation di questo sport fu sorprendente ed in tutto il mondo si scatenò un forte movimento con l’inevitabile inizio di eventi e gare organizzate.

La prima edizione della Roma-Ostia infatti fu svolta nel 1974 con un percorso differente da quello attuale (7km in meno) e con soli 313 atleti classificati.

Tra la fine degli anni '90 e all'inizio degli anni 2000 si è verificato un secondo boom in Europa, caratterizzato da un aumento notevole della partecipazione alle gare organizzate; il tasso di partecipazione è aumentato in media del 300% in tutto il continente Europeo fino a diventare il movimento di massa che oggi conosciamo tutti, adesso per esempio la Roma-Ostia ha una media di circa 7000 iscritti ad edizione e la maratona di New York è diventata di fama e risonanza mondiale.

Sono nate tantissime società sportive per podisti dilettanti e semi-professionisti e, visto il continuo crescere delle adesioni, è diventato di fondamentale importanza gestire il flusso di iscrizioni e richieste attraverso sistemi informativi.

Questa tesi si pone l’obiettivo di strutturare un’analisi completa e di procedere con la prima parte dello sviluppo di un gestionale per le società prima citate.

# Tecnologie e metodologie impiegate

La maggior parte delle operazioni eseguite sul web coinvolgono un client e un server. Il primo è un dispositivo remoto che richiede l’accesso ad un servizio, mentre il secondo interpreta la richiesta del client e gli invia una risposta coerente.

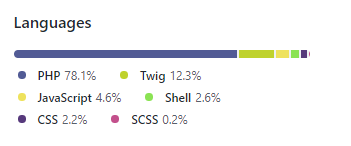
Per creare il gestionale web verrà utilizzato PHP, un linguaggio C-like di scripting server-side, che permetterà l’interpretazione delle richieste del client tramite un web server.

Per permettere un codice ben organizzato, manutenibile e ottimizzato verranno utilizzati anche altri strumenti:

* Symfony, un framework MVC (tra i più usati in php) open source, per la parte relativa al Model si avvale dell’ORM Doctrine.
* Postgres, un DMBS (database management system) ad oggetti che usa la sintassi SQL.
* ORM Doctrine, che permetterà di mappare le entità del database con le classi php.
* Docker, che servirà a containerizzare il database in locale.
* Bootstrap
* Github

Verranno poi utilizzati Javascript e Css per definire nel dettaglio la parte grafica della web-app.

Nello screenshot è presente un riassunto fronito da Github sulle percentuali dei linguaggi usati alla fine dello sviluppo della parte relativa a registrazione, login, e dashboard.



## PHP - Wikipedia2.1 Cos’è PHP

PHP (*Hypertext Preprocessor*) è uno dei più diffusi linguaggi di programmazione open-source generico.

È utilizzato per lo più in ambito web come linguaggio server-side, è stato progettato per pagine web dinamiche integrate perfettamente con i database, può essere utilizzato anche per applicazioni stand-alone con interfacce grafiche o per script a riga di comando.

Nasce nel 1994 da Rasmus Lerdorf, inizialmente come CGI (common gateway interface) binario scritto in C per tracciare il numero di connessioni al suo sito, e dopo 4 anni venne rilasciata la terza versione che, entro la fine del 1998, coprì il 10% dei web server. Attualmente PHP è utilizzato dal 78,9 % dei siti web con un linguaggio server-side conosciuto.

Le caratteristiche principali di questo linguaggio sono:

* Tipizzazione dinamica: la possibilità di non assegnare ad ogni variabile un tipo, infatti il controllo viene effettuato a tempo di runtime e l’interpretazione è automatica.
* Supporto di molti paradigmi: Supporta l’imperativo, il funzionale, il procedurale e l’object-oriented.
* La facilità di effettuare operazioni con le stringhe
* Gli array associativi
* Facilità di accedere alle richieste http di tipo GET e POST
* Accesso in lettura e scrittura dei cookie del browser: I cookies sono parti di testo utilizzate per tracciare informazioni relative ad un sito dal client.
* Supporto alle sessioni sul server: le sessioni sono utilizzate per memorizzare informazioni e preferenze dell’utente, vengono salvate sul server. Utilizzate per esempio nella gestione del log-in, che una volta effettuato rimane valido fino alla fine della sessione senza che l’utente debba effettuarlo per ogni parte del sito protetta per utenti registrati.
* Database: Ci sono molte librerie per accedere alla maggior parte dei database (MySQL, Postgres, SQLite) sia in scrittura che in lettura.
* Multipiattaforma: può essere utilizzato sia in ambienti unix che su windows.

## 2.2 I metodi GET e POST

I client non richiamano direttamente i documenti HTML ma inviano determinati dati con i metodi di richiesta del protocollo http, i più importanti sono Get e Post.

Il primo permette di accodare i dati all’indirizzo della pagina mettendo dopo l’url un punto interrogativo e la coppia nome=valore dei dati interessati, in caso di più variabili esse vengono separate dal segno meno. Gli svantaggi di questi metodi sono ovviamente la totale assenza di protezione dei dati e la sua capacità limitata, infatti nell’url possono essere inseriti massimo 2000 caratteri ed esclusivamente in ASCII.

Il metodo POST scrive i parametri url nella richiesta http ma li nasconde all’utente e qui non c’è un limite di grandezza, lo svantaggio di questo metodo però è che, per esempio, aggiornando la pagina durante la compilazione di un form i parametri vanno nuovamente inseriti e di conseguenza trasmessi un’altra volta, il rischio è di duplicare l’invio dei dati (rischio comunque prevedibile con facili controlli). Al contrario del metodo GET accetta anche valori binari.

In genere si preferisce il metodo POST per inviare dati o file al server, come nel caso di form, mente il metodo GET è preferibile in caso di invio di preferenze e impostazioni per pagine web.

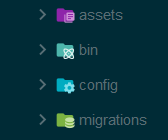
## Symfony, High Performance PHP Framework for Web Development2.3 Symfony Framework

Esistono molti framework per PHP, per esempio Zend, CakePHP, CodeIgniter, eYii, Laravel, ognuno di essi ha caratteristiche diverse, per questo progetto si è scelto di utilizzare Symfony, un framework open source. Al contrario degli altri, Symfony opera ad un livello di astrazione più elevato, orientato alle operazioni CRUD (Create, read, update, delete) permettendo una scalabilità elevatissima.

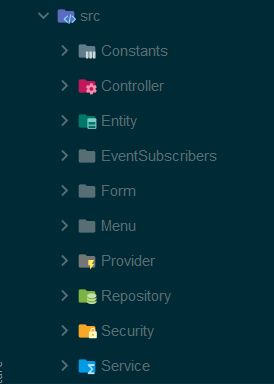
### Il pattern MVC2.3.1 Pattern MVC

Lo scheletro si basa sull’architettura MVC (Model – View – Controller), che separa nettamente la parte grafica (front-end) dalla logica di back-end e dall’accesso ai dati garantendo quindi la totale indipendenza tra le parti, importantissima per la pulizia del codice, per la sua manutenzione e per implementare features in futuro.

Symfony ‘estende’ queste caratteristiche, dividendo in directory tutte le parti del codice in base al loro funzionamento, di seguito le più rilevati per questo progetto:

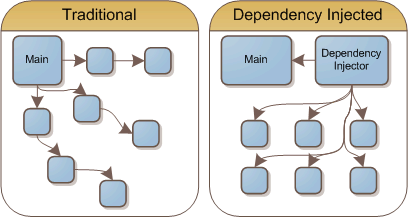
* Assets: contiene i file CSS e JS, necessari allo sviluppo relativo al lato front-end,

che contribuisce a migliorarne lo stile e gestire la logica client-side.

* Config: contiene i file di configurazione dell’ambiente di sviluppo (in genere .yaml) che gestiscono le direttive principali per il routing e la security dell’applicazione, quindi dalle rotte che eseguono i controller fino alle gerarchie dei ruoli.
* Migration: qui ci sono i file che contengono le migrazioni relative al lavoro dell’ORM, infatti creando delle entità o aggiungendo attributi ad una tabella attraverso la console Symfony prepara questi file visualizzabili prima dell’esecuzione che permettono di visualizzare il codice che sta per essere eseguito.
* Src è la cartella in genere più corposa e contiene al suo interno queste sotto-cartelle:
* Controller: contiene tutti i controller della web-app, essi sono composti da funzioni php scritte dall’utente e rispondo ad una *request* restituendo una *response* che può essere una pagina html (o altri formati come XML o json), il download di un file o una pagina di errore. Fondamentalmente gestisce tutte le logiche necessarie per reindirizzare ad una pagina.
* Entity: Contiene i file php per la mappatura delle tabelle del DB, ogni istanza dell’oggetto corrisponde ad un record della tabella relativa. Qui si trovano anche le funzioni di base per accedere all’oggetto, come i setter e i getter.
* Form: all’interno sono presenti dei file che contengono vari ‘form-field’ scritti però in php tipizzato, questo perché la creazione e l’elaborazione dei moduli HTML, dalla convalida all’invio dei dati fino alla mappatura di essi sugli oggetti, è ripetitivo e rischia di indurre ad errori. In questo modo i form risultano più veloci da scrivere e semplificando anche le operazioni più complesse.
* Repository: contiene le classi relative alle entità che si interfacciano con il database, all’interno ci sono i costruttori, gli add e i remove per gli oggetti, oltre a ciò qui vanno inseriti i query builder creati dallo sviluppatore.
* Security: al suo interno si trovano i voter, ovvero i metodi symfony per gesitre i permessi centralizzando quindi tutta la logica al fine di evitare ridondanze.
* Templates: in questa cartella di trovano tutte le viste, in formato Twig, un modello flessibile e rapido nella renderizzazione. Ha una sintassi simile all’HTML ma permette di inserire anche un minimo di logica al suo interno, funzioni come i Get per gli oggetti e Upper per le operazioni con le stringhe, oltre a cicli for.

### 2.3.2 Dependency injection

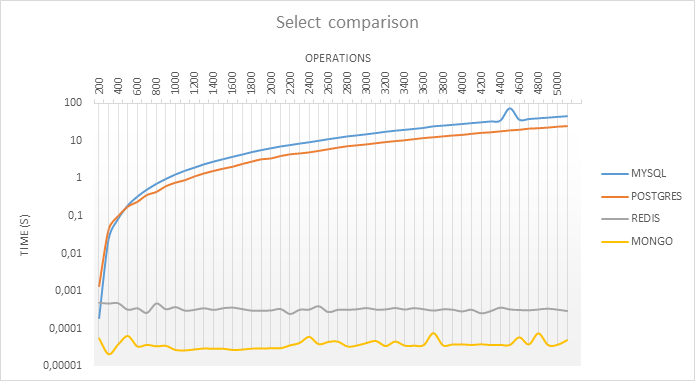
Tra le caratteristiche più importanti di Symfony troviamo la dependency injection, un pttern di programmazione che permette alla classe php di non inizializzare le proprie dipendenze. Una dipendenza è la variabile di una classe e l’injection è l’inizializzazione della stessa, il pattern prima citato permette l’indipendenza tra la classe e la creazione degli oggetti da cui dipende grazie ad un componente separato.



## PostgreSQL Advantages and Disadvantages : Aalpha – Aalpha2.4 Postgres

Postgres (o PostgreSQL) è un sistema open source di database relazionale ad oggetti nato da un progetto dell’università della california a Berkeley. Al contrario di MySQL è completamente conforme a SQL e permette quindi l’utilizzo di tutte le funzioni utili. Oltretutto è il database consigliato da Symfony framework che contiene un bundle ‘separato’ per integrare l’ORM Doctrine.

Dal confronto offerto da ‘Medium’ (una piattaforma di pubblicazione di articoli scientifici online creata dal cofondatore di Twitter) si può notare come postgres, oltre a presentare una compatibilità più elevata con le funzioni SQL, permetta anche una maggiore velocità nell’eseguire Query di interrogazione dei dati.



## 2.5 Doctrine ORM

Un’ORM permettere di gestire tutte le funzioni del database attraverso un’interfaccia orientata agli oggetti, nel nostro caso Doctrine permette di mappare le classi PHP con le tabelle del DB attraverso delle librerie. I vantaggi dell’utilizzo di Doctrine sono molteplici, tra i più importanti troviamo:

* La manipolazione di dati e oggetti, infatti permette di accedere ai dati del DB come oggetti e quindi modificarli senza che lo sviluppatore acceda direttamente a Postgres.
* Consente la validazione dei dati prima di un INSERT o di un UPDATE
* Gestisce le relazioni tra le entità e quindi permette di accedere a tutti gli oggetti collegati con una foreign-key all’oggetto richiesto.
* Permette maggiore velocità di sviluppo rispetto allo scrivere a mano sia la costruzione del database che le classi PHP.
* Riduce la rindonzanza della scrittura del codice per la connessione al database al mapping dei risultati.
* Permette la scalabilità del database senza modifiche importanti alla web-app.

Per le query più complesse, non comprese in Doctrine, è possibile avvalersi del QueryBuilder che permette di scrivere Query da zero attraverso un linguaggio misto SQL all’interno dell’applicazione.

Gli svantaggi dell’utilizzo di un ORM sono pochi, tra i quali si trovano soprattutto problemi per imparare ad utilizzarli, infatti sono moduli molto complessi e l’apprendimento di essi è abbastanza lento rispetto al semplice SQL, oltretutto il rischio è di iniziare ad utilizzare gli ORM senza una perfetta conoscenza di ciò che accade e quindi non riuscire a risolvere i problemi una volta che essi si presentano.

## Bootstrap (front-end framework) - Wikipedia2.6 Bootstrap

Bootstrap è un framework gratuito e open source pensato per il web, in particolare per il front-end. L’obbiettivo principale di questo componente è aiutare gli sviluppatori a creare pagine web responsive senza duplicare inutilmente codice css, esiste in due versioni: precompilato e basato su una versione del codice sorgente.

È molto facile da imparare e ha una struttura semplice basata su HTML, CSS e Javascript, per importarlo bastano poche righe di codice e per usare le sue componenti basta inserire la classe corretta dentro il tag HTML.

## 2.8 Docker

**Docker** è una piattaforma software open source che permette la creazione e la gestione di applicazioni containerizzate, e quindi agli sviluppatori di operare sui container. Un container raccoglie il software in unità standardizzate, docker permette quindi di fornire all’applicazione le risorse per funzionare in qualsiasi ambiente. I principali vantaggi che derivano dal suo utilizzo sono quindi:

* La portabilità, infatti il processo di distribuzione è basato su file di immagine e questo facilita nella condivisione dello sviluppo o del deploy su ambienti diversi.
* La leggerezza, al contrario delle virtual machine i container non includono intere istanze del sistema operativo ma solo i processi e le dipendenze fondamentali per permettere l’esecuzione del codice.

## 2.7 GitHub

Per il versionamento del software è stato utilizzato GitHub, esso serve per tracciare e gestire le modifiche al codice, è utile sia per avere uno storico, e quindi la possibilità di fare revert di modifiche non funzionali e rispristinare vecchie versioni, che per condividere modifiche tra colleghi e di conseguenza gestire i ‘conflitti’, ad esempio nel caso se ne creino c’è la possibilità di accettare singolarmente le modifiche o di effettuare un merge scegliendo riga per riga cosa tenere.

# Requisiti del gestionale

Dopo aver deciso quindi l’ambiente di sviluppo, si procede con la stesura dei requisiti funzionali al fine di comprendere come sviluppare la base dati.

## Requisiti Funzionali

Trattandosi di un gestionale per società sportive dilettantistiche, per prima cosa dobbiamo individuare le funzionalità previste:

Le società sono sviluppate attraverso:

L’amministratore della società, che dovrà avere accesso a tutti gli iscritti, all’inserimento e all’eliminazione o l’approvazione di un nuovo tesserato, oltre a visualizzare gli iscritti alle gare, lo stato dei pagamenti, le gare passate e future e i risultati delle stesse.

Gli iscritti avranno la possibilità di accedere alla loro area personale dove potranno visualizzare e modificare i propri dati personali (compreso l’upload del certificato medico e la possibilità di effettuare il pagamento), vedere i risultati delle precedenti gare e iscriversi alle successive. Oltre a ciò ci sarà una distinzione per le persone affette da disabilità al fine di garantire anche a loro un servizio di prenotazione per le gare adatte.

Alla fine della procedura di iscrizione sarà necessaria la generazione automatica di un report in pdf con le informazioni anagrafiche della persona fisica e la società di appartenenza accessibile dall’amministratore della società, che potrà così procedere ad inoltrare il tutto ai gestori delle varie gare.

Sarà fondamentale la verifica dello stato del certificato medico (consegna e data di scadenza) e dello stato del pagamento della quota (se in corso di validità), in caso mancasse uno dei due requisiti l’atleta potrà comunque effettuare il login per sistemare queste mancanze ma in caso richiedesse l’iscrizione ad una gara il sistema mostrerà un messaggio di errore.

## 3.2 UML

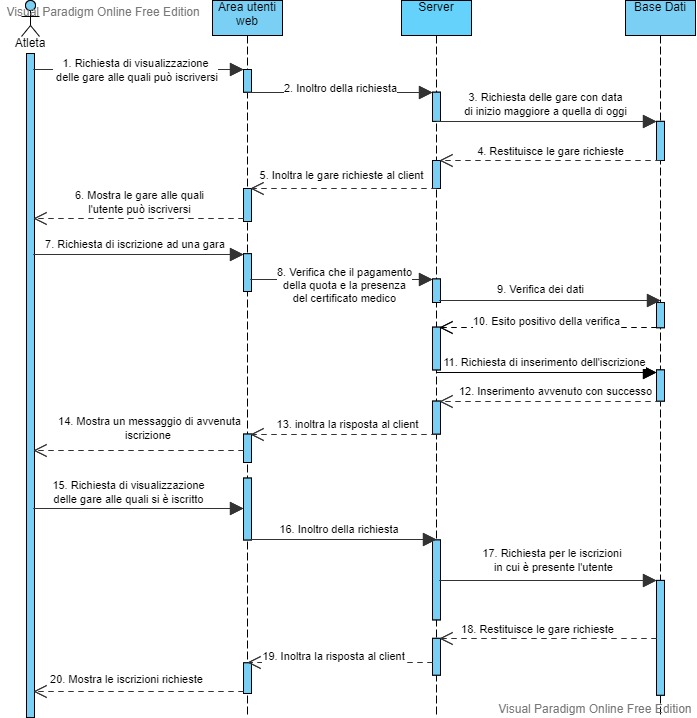
Per strutturare al meglio i requisiti di questo gestionale sarà utile avvalersi ‘UML’ (Unified Modeling Language), un linguaggio nato per unificare approcci differenti e creare quindi uno standard per descrivere soluzioni analitiche e progettuali in modo sintetico e comprensibile.

La funzione fondamentale di questa tipologia di rappresentazione grafica è il creare una gerarchia tra le funzioni e scomporre quindi il software in micro-attività al fine di comprenderne a fondo i requisiti insieme al cliente e, allo stesso tempo, effettuare scelte progettuali e definire priorità di sviluppo.

Per questo progetto si è scelto di rappresentare le scelte progettuali delle attività principali attraverso il ‘sequence diagram’ e lo ‘use case diagram’, questo perché il primo è stato utile al fine di comprendere quali fossero le entità fondamentali, e il secondo per definire i passaggi che l’utente avrebbe dovuto fare per il caso d’uso descritto all’interno del gestionale.

### 3.2.1 Sequence Diagram

Il **sequence diagram** mostra le dinamiche, in ordine di tempo di esecuzione, delle interazioni fra gli oggetti o fra le varie parti del software. In questo caso si è scelto di rappresentare con questo diagramma il caso d’uso di maggior rilevanza, ovvero la richiesta di iscrizione dell’atleta ad una determinata gara:

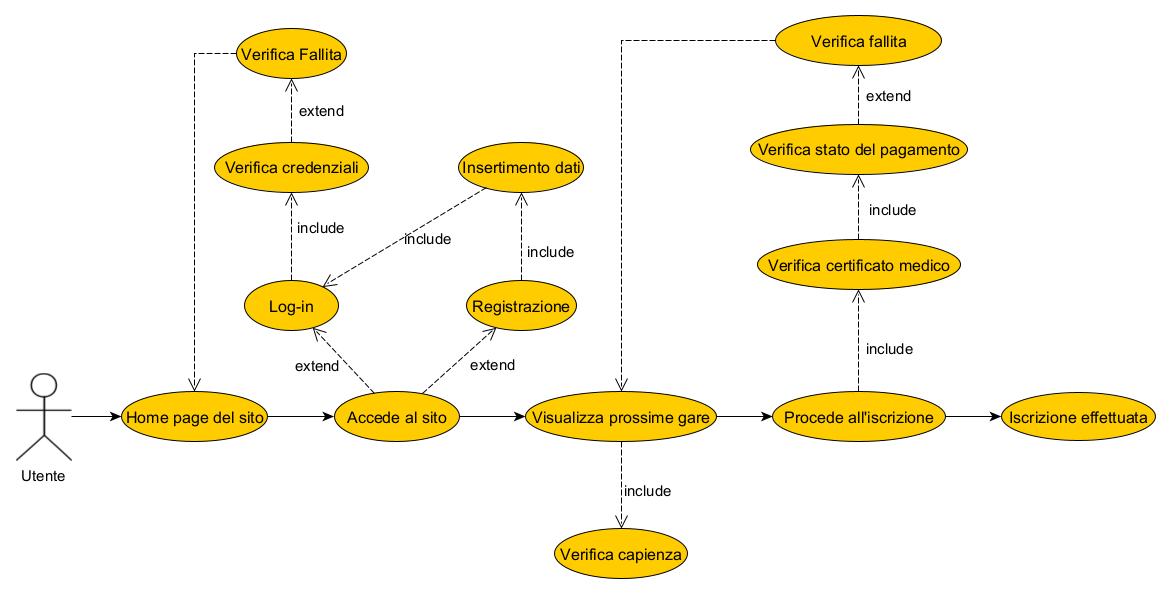


Nelle colonne sono presenti le parti principali del software e l’utente che da inizio alla sequenza attraverso delle richieste, mentre in orizzontale le operazioni effettuate per rispondere alla request.

Per realizzare questo grafico è stato utilizzato Visual Paradigm Online.

### 3.2.2 Use Case Diagram

Questo diagramma si utilizza per descrivere un particolare procedimento all’interno del software dal punto di vista dell’utente, questo permette di ottenere un’idea più chiara dei requisiti del sistema. Nel grafico vengono utilizzati gli ‘actor’ che danno inizio alla sequenza, nel nostro caso si tratta dell’atleta che effettua il log-in alla propria area personale e che vuole prenotarsi per una gara.

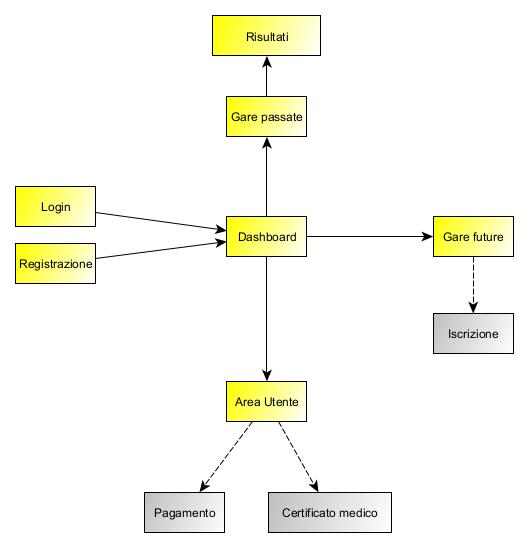
****

In questo grafico si può vedere il flusso relativo alle richieste dell’utente, cioè i passaggi che l’utente deve compiere per registrarsi ad una gara, e il flusso del back-end (verticale) così da stabilire a grandi linee la logica applicativa dell’operazione in esame.

Per questo grafico è stata utilizzata l’app Yed visual graph.

### Flusso del Front-end

Al fine di avere più chiari i passaggi dell’utente all’interno delle pagine navigabili e per garantirne la maggior user-experience possibile è stato fatto un grafico per il flusso della navigazione rispetto alle pagine web:



In giallo le pagine web effettive mentre in grigio, collegate con link tratteggiati, quelle che a seconda delle necessità in fase sviluppo potranno essere semplici pop-up per inserire file (o pagare attraverso API nexi o mastercard) o, come nel caso dell’iscrizione, sezioni apribili della pagina attraverso menù a tendina al fine di effettuare la prenotazione potendo visualizzare e inserire tutti i dati necessari.

Per questo grafico è stata usata l’app Yed visual graph.

Appurati i requisiti fondamentali si può procedere allo sviluppo della base dati.

# Base dati

Come anticipato il database sarà un solo schema Postgres.

La base di dati è stata quindi così pensata:

## Schema E-R

Nella tabella **Personal data** ci saranno tutte le informazioni anagrafiche degli atleti (utenti).

Il certificato medico è fondamentale in un’associazione sportiva quindi è stata creata la tabella **Medical certificate** con la relativa foreign key alla tabella Personal data per permettere il controllo anche sulla scadenza e sulla tipologia, in caso di mancata consegna essa non sarà valorizzata.

Per la gestione dei ruoliè stata creata la tabella **Roles**, al cui interno ogni record sarà un ruolo collegabile all’utente, tra i quali:

* *Atleta*: utenza di base, permessi in lettura e scrittura dei propri dati personali e permessi di lettura dei risultati oltre alla possibilità di iscriversi alle gare.
* *Admin*: utenza che ha il pieno controllo del gestionale, quindi completa visione delle iscrizioni alle gare e dei pagamenti oltre ai certificati medici.

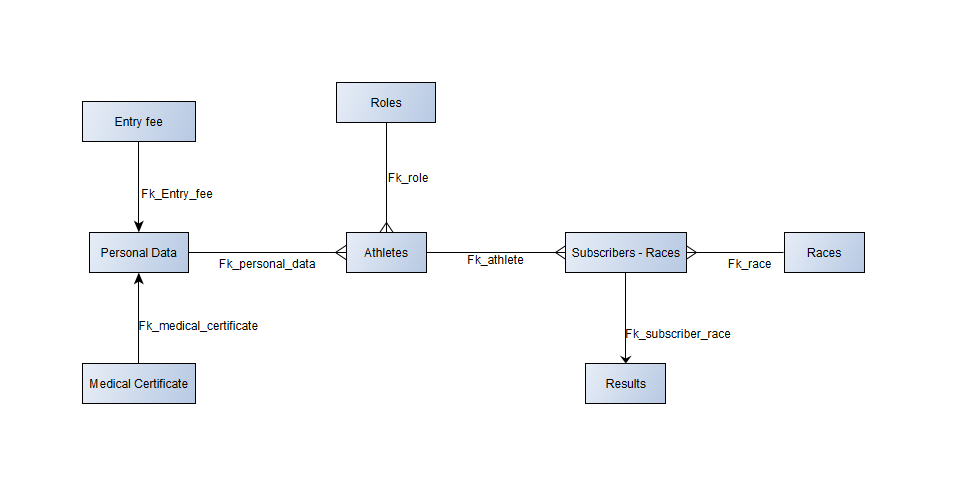
Una persona fisica potrà avere più utenze (in base al ruolo, ad esempio gli amministratori potranno comunque partecipare alle gare con una seconda utenza collegata alla stessa Personal data), perciò è stata aggiunta una tabella chiamata **Athletes** con una foreign key sia dalla tabella Personal Data che dalla tabella Roles.

Laquota associativa verrà gestita al pari del certificato medico, quindi una tabella **Entry fee** collegata alla personal data con la foreign-key a null in caso di mancato pagamento.

Ogni Atleta potrà vedere le gare future e iscriversi in una o più di esse, è stata pensata quindi una tabella molti a molti **Subscribers – Race** chepermetterebbe di avere un id\_iscritto che si riferisce ad un utente iscritto ad una specifica gara.

Per gestire i dati relativi ai risultati di una determinata gara si utilizzerà un’entità di nome **Results** dove si potranno visualizzare i tempi di una determinata gara di un utente collegando l'id iscritto come fk.

Ogni gara sarà un record nella tabella **Races**.



Sarà quindi necessario pensare agli attributi di queste tabelle e quindi sviluppare un diagramma logico.

## 4.2 Schema Logico

La tabella **Personal data** conterrà tutti i dati relativi alla persona fisica che avrà un profilo utente all’interno del gestionale:

* Id
* Nome
* Cognome
* Data di nascita
* Comune di nascita
* Provincia di nascita
* Sesso
* Codice Fiscale
* Data di creazione, quindi un current\_timestamp dal momento dell’inserimento del record
* Flag persone con disabilità, per gestire al meglio le gare adatte.
* Foreign key **Medical Certificate**
* Foreign key **Entry fee**
* Photo, di tipo blob (binary large object)

In questo caso è stato inserito un campo (di default ‘null’) per inserire la foto del utente in codice binario, non direttamente interpretabile dal database, ma per la quale si prevede una funzione di encode e decode al fine di semplificare le operazioni per gli amministrativi e gli utenti.

La foreign key Medical Certificate punterà all’Id del certificato medico consegnato, con rapporto 1:1, in caso mancasse l’attributo sarà non valorizzato, la stessa logica verrà applicata al pagamento della quota con la foreign key Entry fee

Per quanto concerne la gestione dei certificati medici e quindi la relativa Entità **Medical Certificate** si è scelto di gestire solo i dati principali quindi:

* Id
* Data di consegna
* Data di scadenza
* Tipologia, quindi se semplicemente di sana e robusta costituzione o se agonistico.
* File, di tipo blob

Allo stesso modo nella tabella **Entry fee** si troveranno gli attributi:

* Id
* Data di pagamento
* Data di scadenza
* Importo

Per l’entità **Roles**, che verrà trattata come una tipologica,gli attributi saranno:

* Id
* Nome del ruolo
* Codice (es. ‘ROLE\_USER’, ‘ROLE\_ADMIN)

L’entità **Athletes** servirà per le gestione degli utenti, per questo al suo interno ci saranno gli attributi utili alla gestione del log-in oltre alle foreign\_key verso le entità prima citate:

* Id
* Indirizzo email (che verrà usato anche come ‘nome utente’)
* Password (alla quale verrà effettuato un hash per motivi di sicurezza)
* Data di creazione
* Numero di telefono
* Foreign key **Personal data**
* Foreign key **Roles**

Entrambi i collegamenti con le altre entità permetteranno di associare più utenti ad un singolo record delle relative tabelle.

Per quanto concerne la gestione delle gare nell’entità **Races** gli attributi saranno i seguenti:

* Id
* Nome della gara
* Capienza massima, così da evitare di andare in overbooking con le richieste, in caso la gara non preveda una capienza massima basterà non valorizzarlo
* Data della gara, sarà un Date time che conterrà anche l’orario.
* Tipo di gara (es. ‘Staffetta’, ‘Maratona’, ‘Centro metri’)
* Lunghezza
* Lunghezza effettiva, spesso differente da quella pubblicizzata
* Costo della gara (se presente)
* Indirizzo del punto di ritrovo
* Flag gara sociale, in caso si tratti di una gara esclusiva per i soci avrà valore = 1
* Flag per persone con disabilità, il valore sarà = 1 quando la gara sarà esclusivamente per persone con disabilità o in caso ci siano misure per permetterne la sicura partecipazione (ad esempio la partenza anticipata o avvicinata al traguardo), ciò sarà adeguatamente specificato nella descrizione della gara.

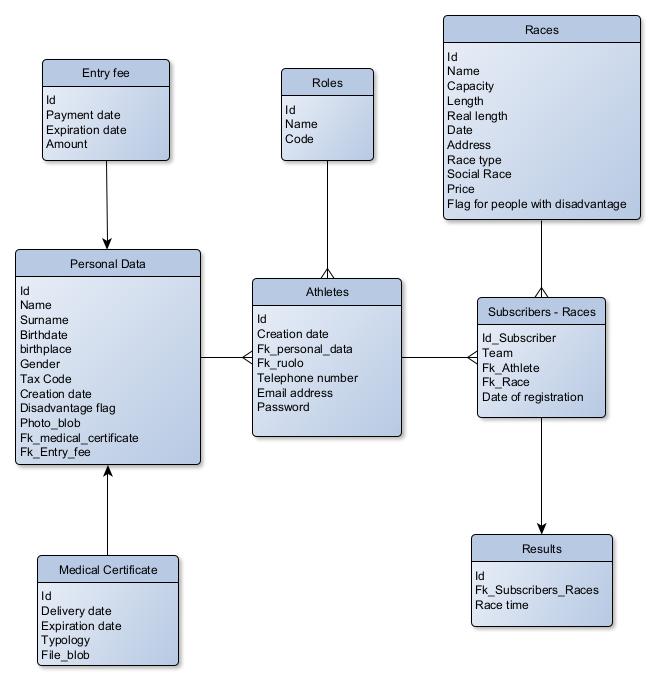
La tabella molti a molti **Subscribers-Races** sarà trattata da entità in modo da avere un id iscritto e una data di iscrizione al suo interno:

* Id
* Team (in caso di staffette servirà per capire il gruppo di appartenenza, in caso di altri tipi di gare individuali basterà non valorizzare l’attributo)
* Foreign Key **Athlete**
* Foreign Key **Race**
* Data di iscrizione

Questa tabella avrà un collegamento alla **Results**:

* Id
* Tempo di gara
* Foreign Key **Subscribers – Race**

All’interno dell’applicazione e quindi nella base dati i nomi degli attributi saranno in inglese per mantenere la coerenza di linguaggio con le funzioni, di seguito il diagramma logico sviluppato.



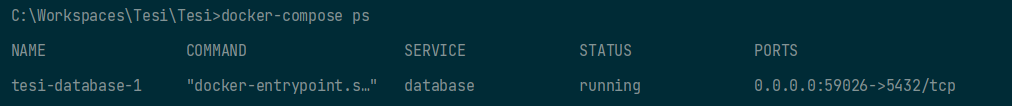
Altri attributi potrebbero essere aggiunti in seguito a seconda delle richieste delle società, doctrine infatti permette modifiche al database in maniera dinamica anche a fronte di un applicativo già sviluppato, che potrà quindi essere personalizzato in base alle esigenze senza impattarne la stabilità.

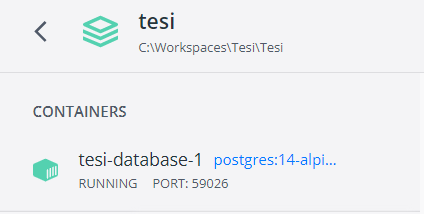
## 4.3 Creazione del database in locale

Per operare all’interno del database in locale sarà utilizzato un container docker attraverso il supporto nativo che symfony offre, infatti, dopo aver creato il progetto, il framework crea due file yaml (docker-compose.override.yml e docker-compose.yml) con all’interno le configurazioni per la virtualizzazione del DB in docker.

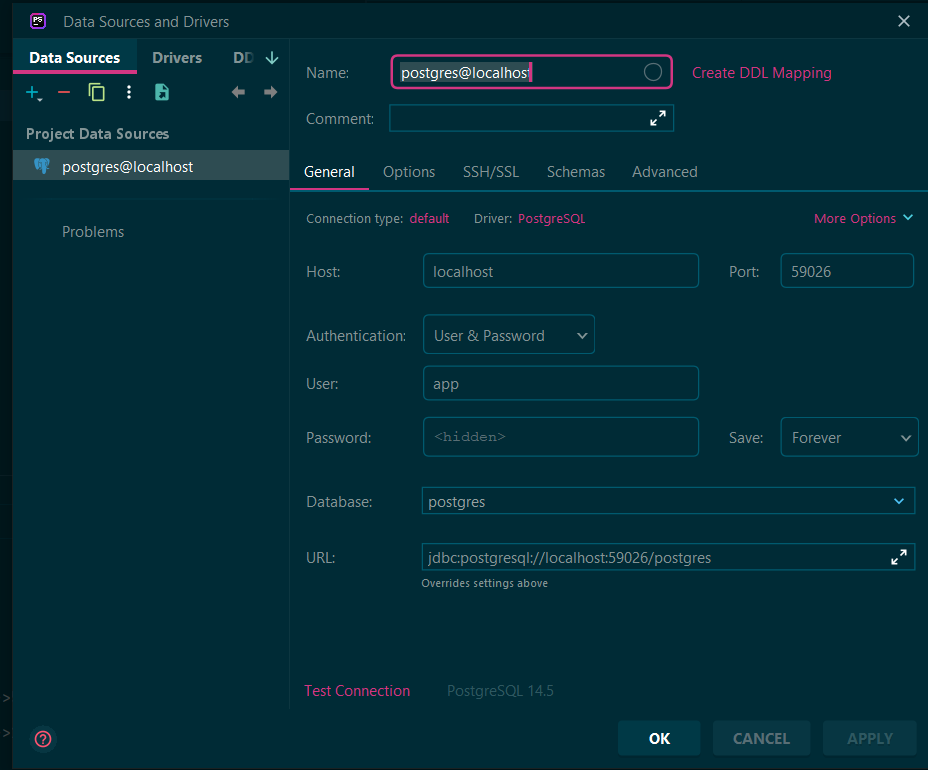
Basta lanciare il comando dal terminale ‘docker-compose up -d’ per avviare il container che permette la containerizzazione della base dati, attraverso questo comando docker crea e avvia il container lasciandolo in esecuzione in background.

Per accertarci della riuscita dell’operazione si può utilizzare ‘*docker-compose ps*’ attraverso la console, che restituisce lo stato dei servizi attivi:





La stessa cosa si può verificare anche attraverso l’app desktop di Docker:



Una volta accertata la presenza del container in stato di ‘running’ si può effettuare la connessione a php storm, così da avere il database e la relativa console all’interno dell’IDE come in figura.

Nel prima citato file ‘docker-compose.yml’ potremmo inserire il nome utente e la password, mentre nel file ‘docker-compose.override.yml’ il numero di porta che Docker dovrà mappare.

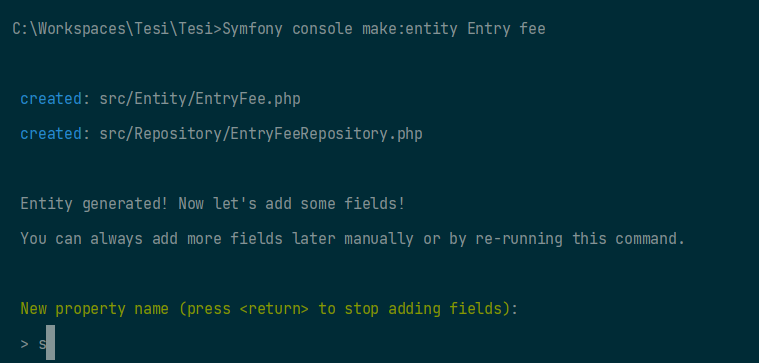
La porta da inserire sarà quella mappata da Docker (59026 nel nostro caso) e non quella scelta.

# Sviluppo della web app

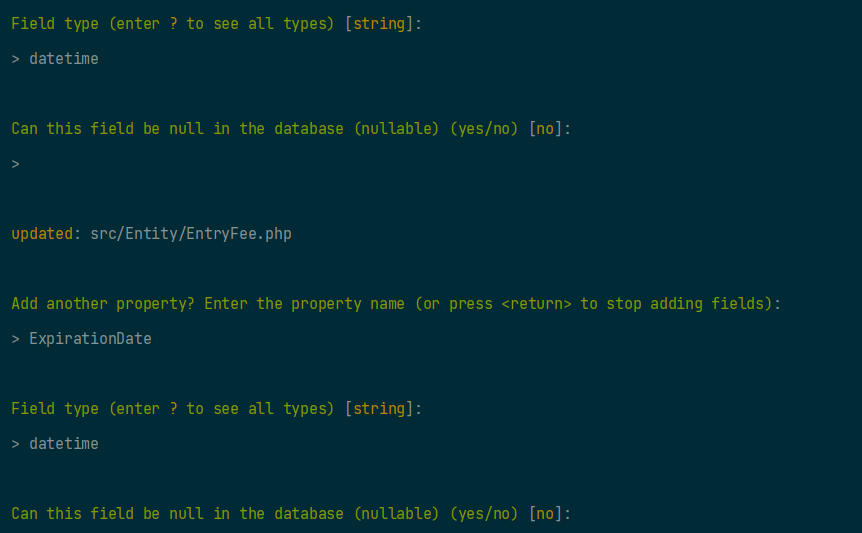
Si può procedere allo sviluppo vero e proprio del gestionale, per prima cosa quindi si procede alla creazione dello schema del database che conterrà le tabelle prima citate.

## 5.1 Creazione delle tabelle

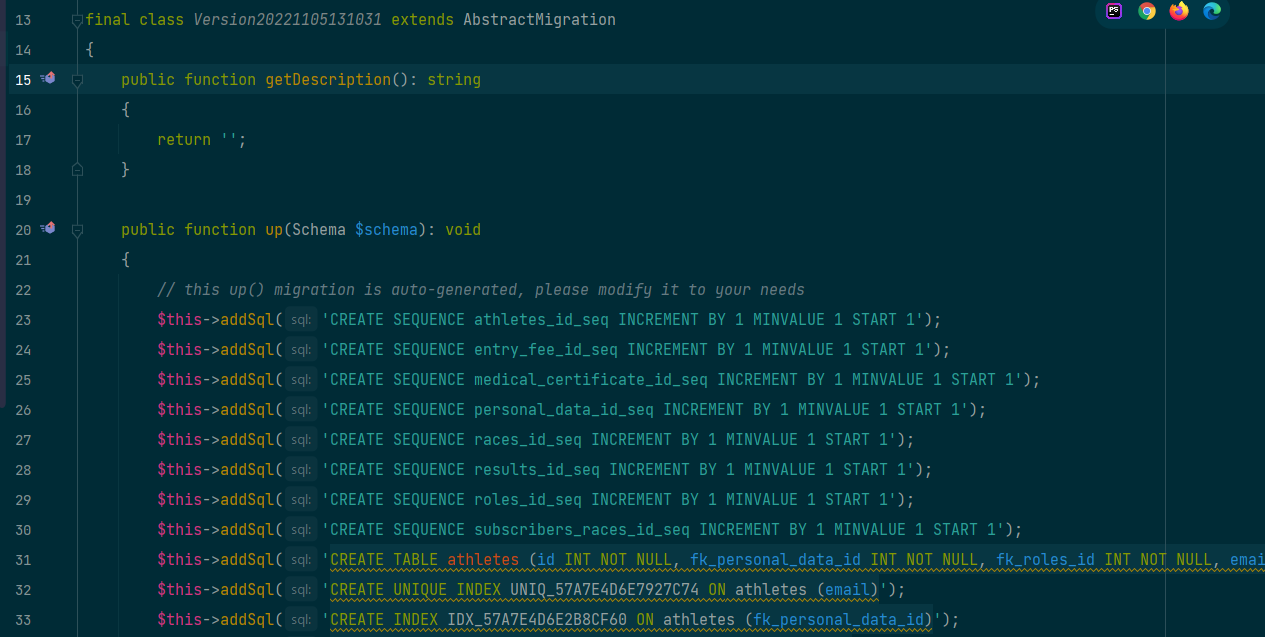
Grazie a Doctrine per creare l’entità non è necessario scrivere i vari ‘create table’ ma è sufficiente scrivere in console ‘*Symfony console make:entity NomeEntità’* (Entry Fee nell’esempio)



Il terminale interroga l’utente sulle proprietà (attributi) da aggiungere per poi specificarne il tipo, ad esempio per l’entità Entry Fee:



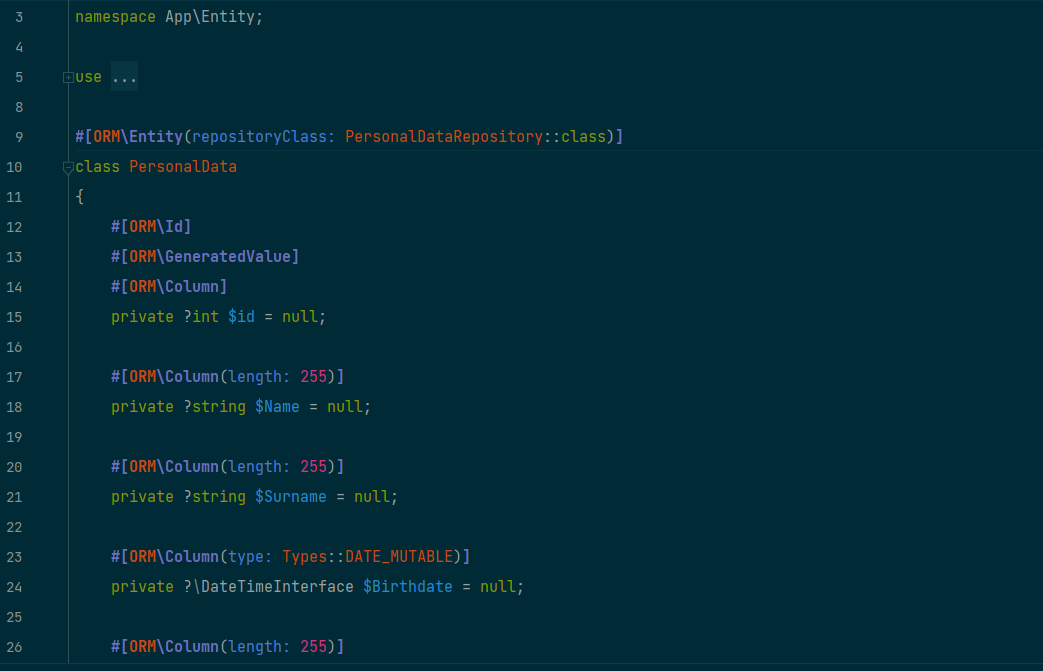
Per comunicare la fine del processo di creazione delle entità e quindi per inizializzare la migrazione si scrive in console ‘*Symfony console make:migration*’ che permette all’ORM di generare un file di migrazione in php che crea effettivamente le tabelle in linguaggio SQL:



Una volta accertati della correttezza dei dati nel file, con il comando ‘*doctrine:migration:migrate*’ si lancia la migrazione.

## 5.2 Creazione delle classi PHP

Il framework durante questo procedimento ha creato anche le classi in PHP, già mappate tramite l’ORM ,e i setter e i getter.



Sono stati aggiunte due funzioni per aggiungere e rimuovere utenti dalla classe personal data al fine di poter gestire l’entità Atleti direttamente dagli oggetti derivanti dalla classe anagrafica:

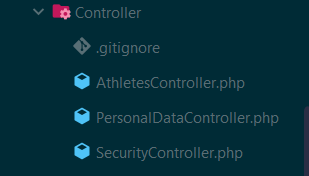


## 5.3 Creazione dei repository

Symfony crea in automatico anche i repository, i metodi find, il relativo costruttore e le funzioni *save e remove*.

Tutte queste funzioni verranno utilizzate all’interno dei controller per gestire tutti i dati del database senza la necessità di creare connessioni a Postgres.

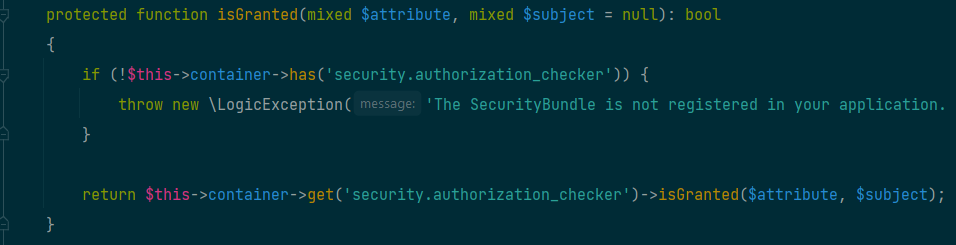
## 5.4 I primi Controller



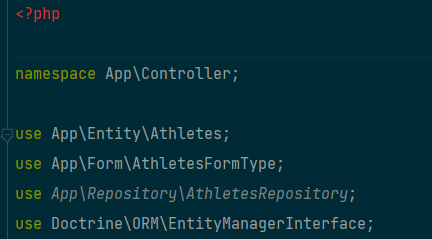
I primi controller che verranno creati sono quelli relativi a all’utente (Atleta), alla persona fisica e alla sicurezza, infatti essi sono necessari per permettere sia la registrazione che il log-in, che sono le prime parti che si è scelto di sviluppare.

Ogni controller deve ‘estendere’ il controller astratto, una classe php generata da symfony che contiene la dichiarazione delle funzioni di base, ad esempio:

* *redirect,* che serve all’indirizzamento verso una pagina specifica o una rotta.
* *renderForm* serve per chiamare un form precedentemente creato.
* *getUser* restituisce l’utente in sessione in quel momento.
* *isGranted* permette di verificare il ‘ruolo’ della sicurezza dell’utente inserendo il grado minimo per l’accesso al contenuto, come si può vedere dallo screenshot:

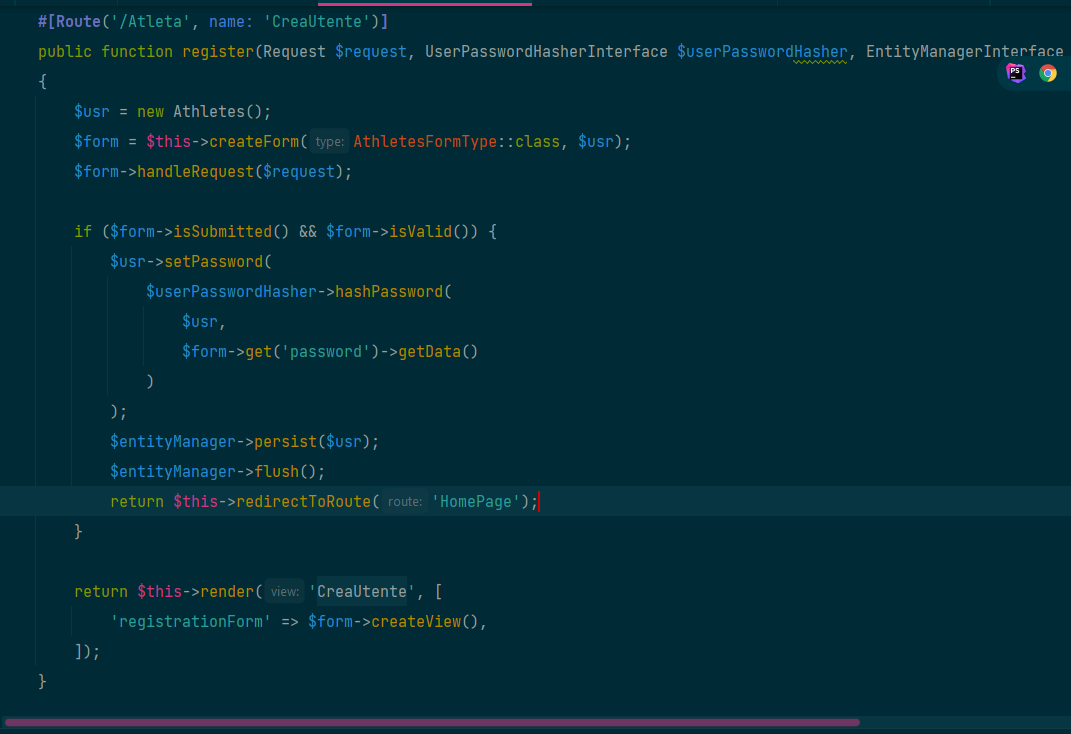


### 5.4.1 Athletes Controller



Dopo aver dichiarato il *namespace* e le dipendenze, è stata creata la classe AthletesController al cui interno verranno inserite tutte le funzioni inerenti all’utente:

La prima funzione creata è stata quella relativa alla registrazione, dopo aver definito la rotta e il nome (register) sono stati inseriti gli argomenti UserPasswordHasherInterface e EntityManagerInterface, il primo serve per gestire l’hashing della password nel server prima dell’inserimento del dato nel database, il secondo per gestire l’entità atleta come una classe e per richiamare quindi le funzioni del repository.



Dopo aver istanziato un nuovo oggetto Atleta è stato richiamato il form ed è stato effettuato un handlerequest del form per gestirne il contenuto una volta premuto il tasto con valore di *submit.* Dopo il controllo relativo alla validità del form, che in caso di esito negativo ricrea il form, viene ‘presa’ la password e viene effettuato l’hashing per poi effettuare il flush per inserire i dati nel db. Infine l’utente viene reindirizzato alla propria dashboard.

La rotta ‘CreaUtente’ è necessario che venga richiamata per iniziare la sequenza di istruzioni, ciò non avviene dal tasto di registrazione ma dal controller della persona fisica, questo perché onde evitare di creare 2 form distinti per le due entità il form precedentemente illustrato è stato annidato all’interno di quello per i dati anagrafici così che l’utente possa compilare il tutto in unica pagina.

### 5.4.2 Personal Data Controller

Esattamente come nel caso precedente sono stati dichiarati sia il namespace che le dipendenze, qui la prima funzione creata è stata quella di *Indexing* che semplicemente porta chiunque si connetta al sito per la prima volta nella HomePage, a prescindere dal link usato.

Per la registrazione è stato richiamato il form relativo alla persona fisica:



Al suo interno contiene anche il form relativo all’utente come prima anticipato:  


Per il salvataggio il funzionamento è lo stesso dell’utente, infatti attraverso un flush i dati inseriti vengono inseriti nel database.

## 5.5 Front-end

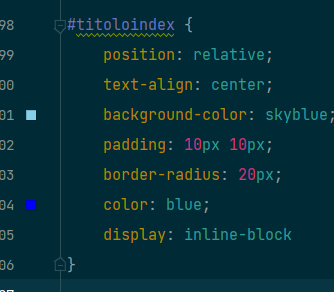
Per quanto riguarda la parte client symfony, come anticipato, si avvale dei twig, dei template che permettono di fare un render html con l’aggiunta di logica php con una sintassi semplificata.

Una sua grande caratteristica è l’ereditarietà, infatti consente di creare dei template ‘base’ e di creare nuove pagine che li estendono, riducendo la ridondanza del codice e aumentando la velocità di sviluppo.

La sintassi è particolare e si divide in due principali rami:

* Le variabili, che devono essere ‘inviate’ dai controller php, alle quali si possono applicare tutte le funzioni di get dichiarate precedentemente all’interno delle classi php che vengono richiamate all’interno di una doppia graffa, ad esempio : {{ Athletes.email}} stamperà a schermo l’email dell’utente valorizzato dal controller.
* La logica, come if o cicli for, che vanno dichiarati all’interno di parentesi graffe con un simbolo di percentuale, ad esempio : {% for athlete in Athletes %} creerà un ciclo all’interno del quale verranno inseriti tutti gli utenti passati dal controller, che quindi dovrà istanziare un’array collection di utenti.

Per lo stile invece è stato utilizzato il CSS, Cascading Style Sheets, che permette la modellazione dei tag HTML attraverso id e classi, il primo per un elemento specifico mentre il secondo viene in genere utilizzato per gruppi di tag.

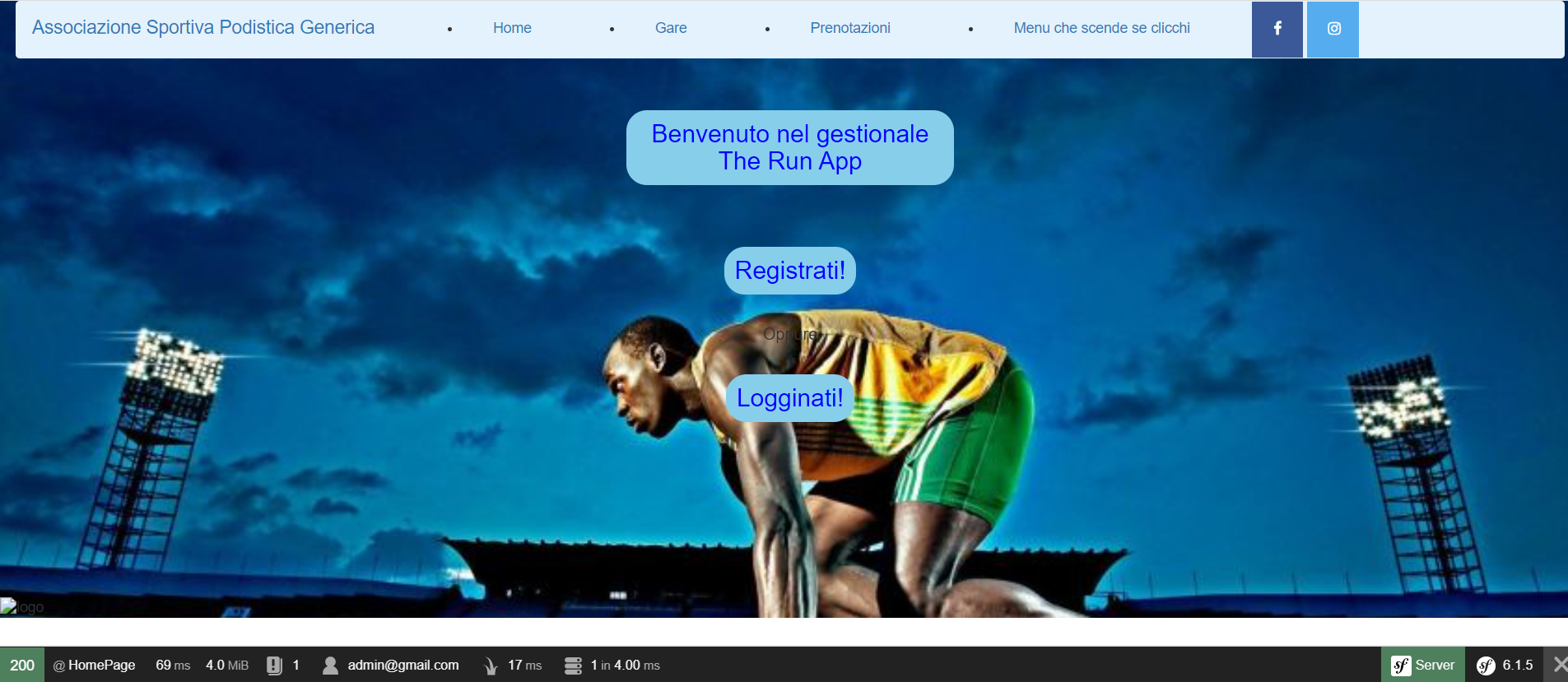


Ad esempio per il titolo nella home page si è scelto di utilizzare questo tipo di stile:

 Che ha generato questo titolo:

## 5.5.2 La grafica dell’app e la navigazione

Questa parte va completata in seguito (max lunedì) in quanto il frontend non è pronto.



Questa è la situazione attuale

aggiungere

Navigazione

Mettere home page

Poi login + registrazione

Infine dashboard

# To do – Future implementazioni

Lo sviluppo per questa tesi si ferma alla Dashboard ma si prevede il rilascio di una beta completa entro pochi mesi. Oltre a ciò che manca rispetto all’analisi (la prenotazione delle gare, i risultati delle precedenti, lo stato del certificato medico e della quota) grande attenzione va rivolta alla reportistica ad uso degli amministratori, infatti essa sarà fondamentale per semplificare l’iscrizione degli atleti alle gare.



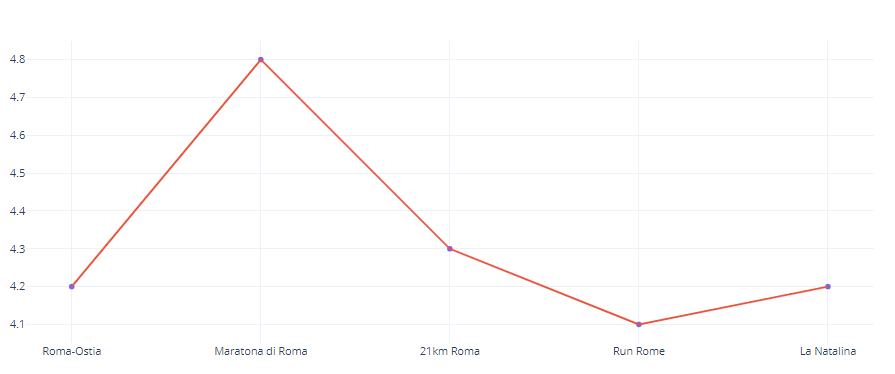
Per permetterne la corretta iscrizione sarà implementato un sistema di calcolo che, in base alla data di nascita, inserirà ogni atleta nella categoria corretta, fino ai 23 anni gli scatti di categoria hanno un età variabile per poi raggruppare ogni 5 anni come in foto, fino alla categoria SM/SF95 che conterrà gli atleti dai 95 ai 100 anni.

All’interno di un file pdf, generato automaticamente dopo la richiesta del client, verranno inseriti tutti gli atleti che hanno effettuato correttamente l’iscrizione alla gara, con il pagamento e il certificato medico valido, raggruppati per gara scelta con i relativi dati anagrafici in ogni colonna, di seguito un facsimile in pseudocodice con sintassi twig:



Il pdf verrà creato con twig grazie al supporto nativo che offre per l’esportazione, di conseguenza basterà creare una pagina web html per poi generare un pdf scaricabile.

Verrà anche dedicata, nella pagina relativa alle precedenti gare e quindi ai risultati, una piccola dashboard per la visualizzazione di grafici dinamici creati con la libreria Plotly di Javascript, di seguito un esempio con le principali gare di Roma:



Il controller effettuerebbe così un calcolo sulla media per ogni km per ogni maratona per poi rappresentare graficamente il tutto, dove un valore più basso equivale una performance migliore.

# Test

Prima del rilascio in test per piccole società verranno svolti dei test interni nei quali, oltre a rispettare i requisiti di accessibilità come da AGID (agenzia per l’Italia digitale), si verificherà che ogni form abbia le giusti convalide obbligatorie dei campi, che i dati nel database rispecchino fedelmente quelli inseriti dall’utente, che il time-out della sessione funzioni in maniera coerente, verrà poi testato il front-end sui browser più conosciuti al fine di evitare problematiche di incompatibilità che renderebbero la user experience meno piacevole, e in ultima analisi che le prestazioni permettano una navigazione fluida e rapida.

Verrà inizialmente chiesto a piccole società, con un massimo di 500 iscritti previsti, di testare il gestionale gratuitamente, a patto di notificare ogni bug e di effettuare un ‘report’ con le migliorie che più apprezzerebbero, per poi rilasciare, in una fase 2, una effettiva beta per testarne l’affidabilità su più grandi moli di dati con un periodo di prova che consentirà poi di effettuare un abbonamento su base mensile per continuare ad usufruire del servizio.

Come alternativa a questo tipo di abbonamento potrebbe essere implementato uno spazio dedicato a dei banner pubblicitari, che potranno essere noleggiati da eventuali partner come negozi di articoli sportivi e centri medici specializzati per consentire visite mediche scontate ai soci.

# Deploy

Per la distribuzione della web app verrà utilizzato Amazon Web Service (AWS), un servizio cloud di proprietà di Amazon estremamente scalabile.

I servizi forniti da AWS sono molti, quelli di interesse in questo caso sono i servizi di elaborazione, tra i quali si trovano EC2, Lambda e Elastic Beanstalk ad esempio, i servizi di archiviazione per il database, quelli di Networking e quelli di messaggistica per la mail necessaria all’attivazione dell’account.

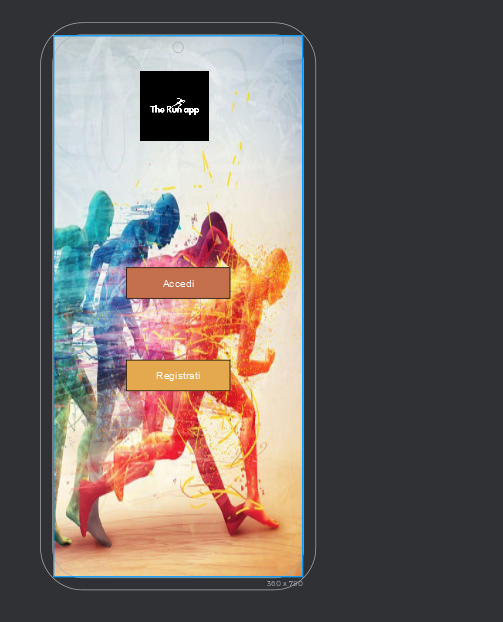
La configurazione scelta sarà di tipo PaaS, Platform as a Service, che permetterà di non preoccuparsi dell’installazione del sistema operativo e del web server, infatti basterà distribuire un container che verrà eseguito in automatico.

Per l’elaborazione è stato scelto Elastic Beanstalk, verrà creato l’ambiente attraverso il composer che installerà tutte le dipendenze necessarie ad eseguire il container.

Per il database verrà usato Amazon Aurora, un RDBMS completamente compatibile con Postgres, per le prime fasi di test gli verrà dedicato uno spazio di dimensioni ridotte per poi incrementarle in caso di bisogno durante la messa in produzione vera e propria.

# Mobile App

Anche qui manca la grafica, questa è da rifare



# Sitografia e bibliografia:

Repubblica: [https://www.repubblica.it/sport/running/storie](https://www.repubblica.it/sport/running/storie/2018/10/05/news/cosa_e_restato_di_quegli_anni_ottanta_viaggio_nella_trasformazione_del_podismo_amatoriale-208190153/)

Wikidata: [https://www.wikidata.org/](https://www.wikidata.org/wiki/Q7380009)

Roma Ostia: <https://www.romaostia.it/la-storia-2/>

Html: https://www.html.it/

W3T : <https://w3techs.com/>

Symfony, documentazione ufficiale: <https://symfony.com/doc/>

Bootstrap, documentazione ufficiale: <https://getbootstrap.com/>

Wikipedia: <https://en.wikipedia.org/wiki/>

Devexp: <https://devexp.io/language/it/2019/04/16/cose-la-dependency-injection-e-come-puo-migliorare-il-design-di-un-progetto-codici-ed-esempi/>

Kinsta: <https://kinsta.com/it/knowledgebase/cosa-e-github/>

Witekio: https://witekio.com/de/

Medium: <https://medium.com>

Kathleen Kavanagh Chapman, Joe Leben. DB2: principi, progettazione e programmazione; 1990

Enrico Zimuel, [Sviluppare in PHP 7. Realizzare applicazioni Web e API professionali](https://amzn.to/3a6PbGp)

Prettyman Steve, php 8, guida allo sviluppo web. 2022

James F. Kurose, Keith W. Ross, Reti di calcolatori e internet. Un approccio top-down; 2017

William Stallings, Sicurezza dei computer e delle reti, 2022.

# Ringraziamenti