

**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 di brutta, utilizzato per gli appunti)

Abstract

Introduzione

**Tecnologie e metodologie impiegate**

1. PHP

1.1 Cos’è PHP? + breve storia del linguaggio

1.2 Caratteristiche e vantaggi

1.3 I metodi GET e POST

2. Postgres e ORM

2.2 Perché usare un ORM

3 Symfony

3.1 Cos’é Symfony

3.2 Perché usare Symfony

3.2 Bootstrap --

IV

4. Progetto

4.1 Requisiti Funzionali

4.2 Casi d’uso

3.4 Studio del database

3.4.2 Schema E-R

3.4.3 Da schema E-R a progetto logico

3.4.4. Implementazione attraverso l’ORM

4.1 Parte relativa al codice e quindi tutto lo sviluppo in symfony

4.1.1 controller per il login e hash della pwd

4.1.2

Conclusioni e sviluppi futuri (classifiche interne, fine sviluppo, ETL)

Bibliografia

Sommario

[***Titolo Tesi*** 1](#_Toc118635522)

[ANALISI DEI REQUISITI E DELLA FATTIBILITÀ DI UNA WEB APP PER LA COMPLETA GESTIONE DI SOCIETÀ PODISTICHE AMATORIALI ​ 1](#_Toc118635523)

[1. Introduzione 5](#_Toc118635524)

[2. Tecnologie e metodologie impiegate 6](#_Toc118635525)

[2.1 Cos’è PHP 7](#_Toc118635526)

[2.2 I metodi GET e POST 8](#_Toc118635527)

[2.3 Postgres 8](#_Toc118635528)

[2.4 ORM 9](#_Toc118635529)

[2.5 SYMFONY Framework 9](#_Toc118635530)

[2.5.1 Pattern MVC 10](#_Toc118635531)

[2.5.2 Dependency injection 11](#_Toc118635532)

[2.6 Bootstrap 12](#_Toc118635533)

[2.7 GitHub 12](#_Toc118635534)

[2.8 Docker 12](#_Toc118635535)

[3. Requisiti del gestionale 13](#_Toc118635536)

[3.1 Requisiti Funzionali 14](#_Toc118635537)

[3.2 UML 14](#_Toc118635538)

[3.2.1 Sequence Diagram 14](#_Toc118635539)

[3.2.2 Use Case Diagram 15](#_Toc118635540)

[4. Base dati 16](#_Toc118635541)

[4.1 Schema E-R 16](#_Toc118635542)

[4.2 Schema Logico 18](#_Toc118635543)

[4.3 Creazione del database in locale 21](#_Toc118635544)

[5. Sviluppo della web app 23](#_Toc118635545)

[5.1 Creazione delle tabelle 23](#_Toc118635546)

[5.2 Creazione delle classi PHP 24](#_Toc118635547)

[5.3 Creazione dei repository 25](#_Toc118635548)

[10. Bibliografia e sitografia : 26](#_Toc118635549)

**Abstract**

# 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.

## 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.

## PostgreSQL Advantages and Disadvantages : Aalpha – Aalpha2.3 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.

## 2.4 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:

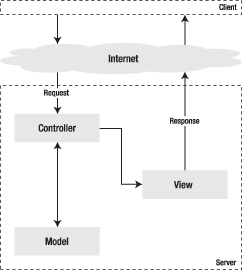
* La manipolazione di dati e oggetti, infatti permette di accedere ai dati del DB come oggetti e quindi modificarli senza che lo sviluppatore acceda direttamente a Postgres.
* Consente la validazione dei dati prima di un INSERT o di un UPDATE
* Gestisce le relazioni tra le entità e quindi permette di 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.

## Symfony, High Performance PHP Framework for Web Development2.5 Symfony Framework

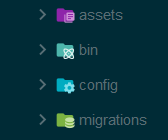
Esistono molti framework per PHP, per esempio Zend, CakePHP, CodeIgniter, eYii, Laravel, ognuno di essi ha caratteristiche diverse, per questo progetto si è scelto di utilizzare Symfony, un framework open source. Al contrario degli altri, Symfony opera ad un livello di astrazione più elevato, orientato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



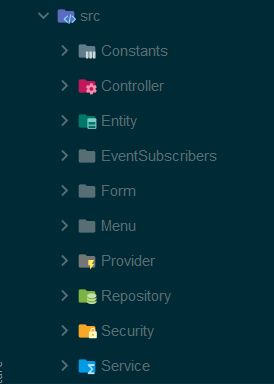
### 2.5.1 Pattern MVC

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.
* Public in genere gestisce le pagine di Index.php.
* 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.

### 2.5.2 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.

## 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.



## 2.7 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.

## 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.

# 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.

## 3.1 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.

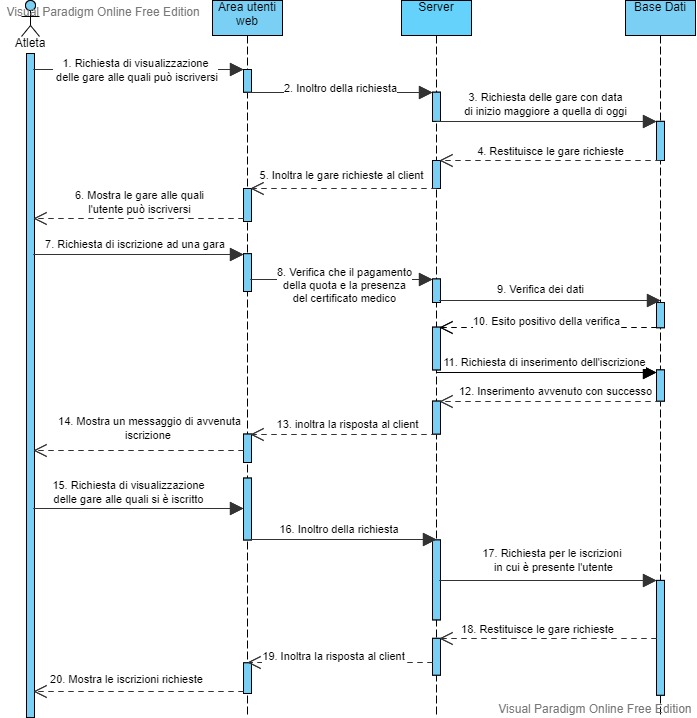
Sarà fondamentale la verifica dello stato del certificato medico (consegna e data di scadenza) e dello stato del pagamento della quota (se in corso di validità), in caso mancasse uno dei due requisiti l’atleta potrà comunque effettuare il login per sistemare queste mancanze ma in caso richiedesse l’iscrizione 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. Per questo progetto si è scelto di rappresentare le scelte progettuali delle attività principali attraverso il ‘sequence diagram’ e lo ‘use case diagram’.

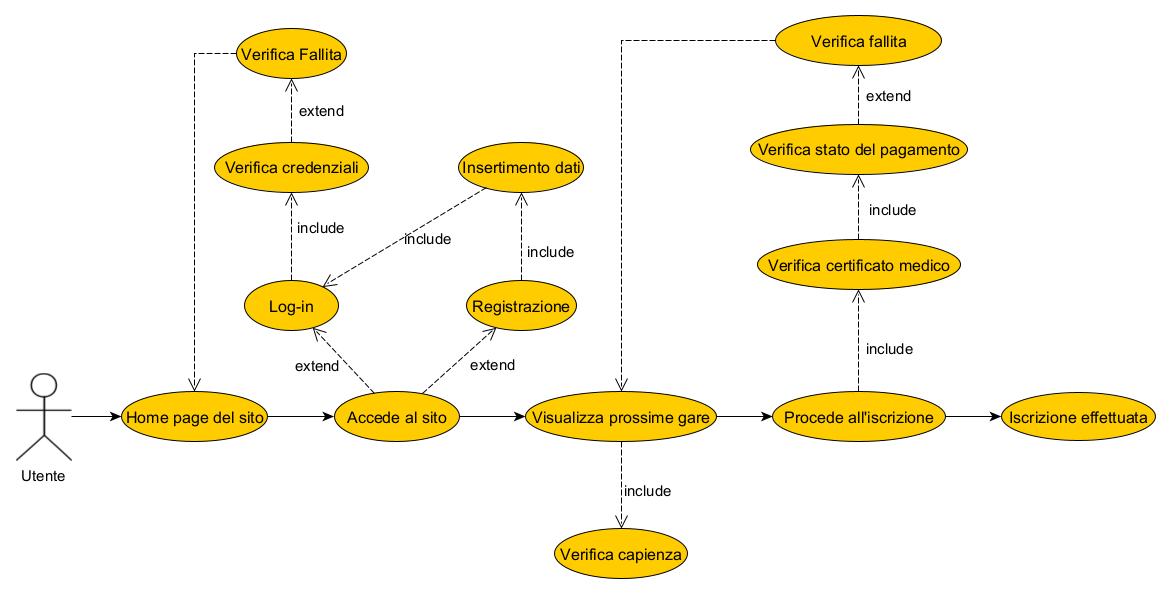
### 3.2.1 Sequence Diagram

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:



### 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.

****

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

# Base dati

La base di dati è stata quindi così pensata:

## 4.1 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.

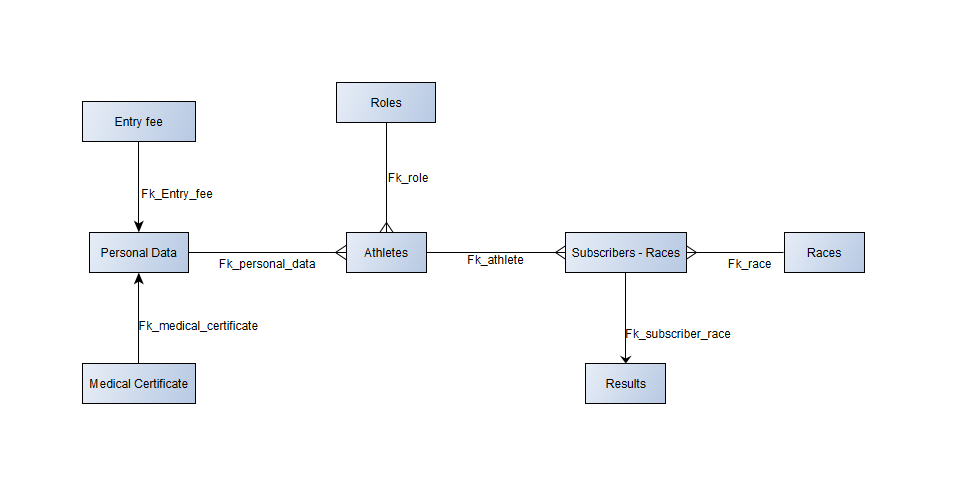
Una persona fisica potrà avere più utenze (in base al ruolo, ad esempio gli amministratori potranno comunque partecipare alle gare con una seconda utenza collegata alla stessa Personal data), perciò è stata aggiunta una tabella chiamata **Athletes** con una foreign key sia dalla tabella Personal Data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

La foreign key punterà all’Id del certificato medico consegnato, con rapporto 1:1, in caso mancasse l’attributo sarà non valorizzato.

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.

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

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

Per l’entità **Roles** 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’)
* Flag per persone con disabilità, il valore sarà = 1 quando la gara sarà esclusivamente per persone con disabilità.

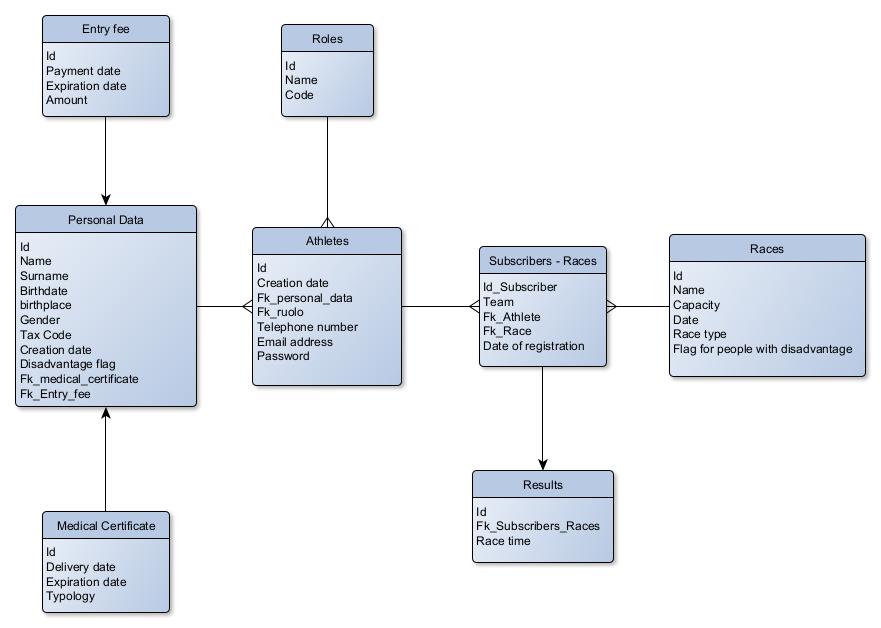
La tabella molti a molti **Subscribers-Races** sarà trattata da entità in modo da avere un id iscritto e una data di iscrizione al suo interno:

* 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.



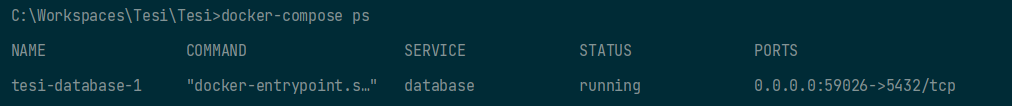
## 4.3 Creazione del database in locale

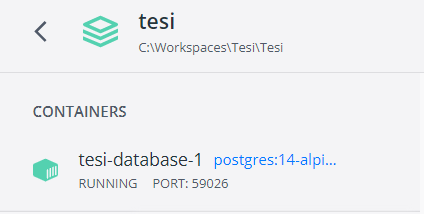
Per operare all’interno del database in locale sarà utilizzato un container docker attraverso il supporto nativo che symfony offre, infatti, dopo aver creato il progetto, il framework crea due file .yml (docker-compose.override.yml e docker-compose.yml) con all’interno le configurazioni per la virtualizzazione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

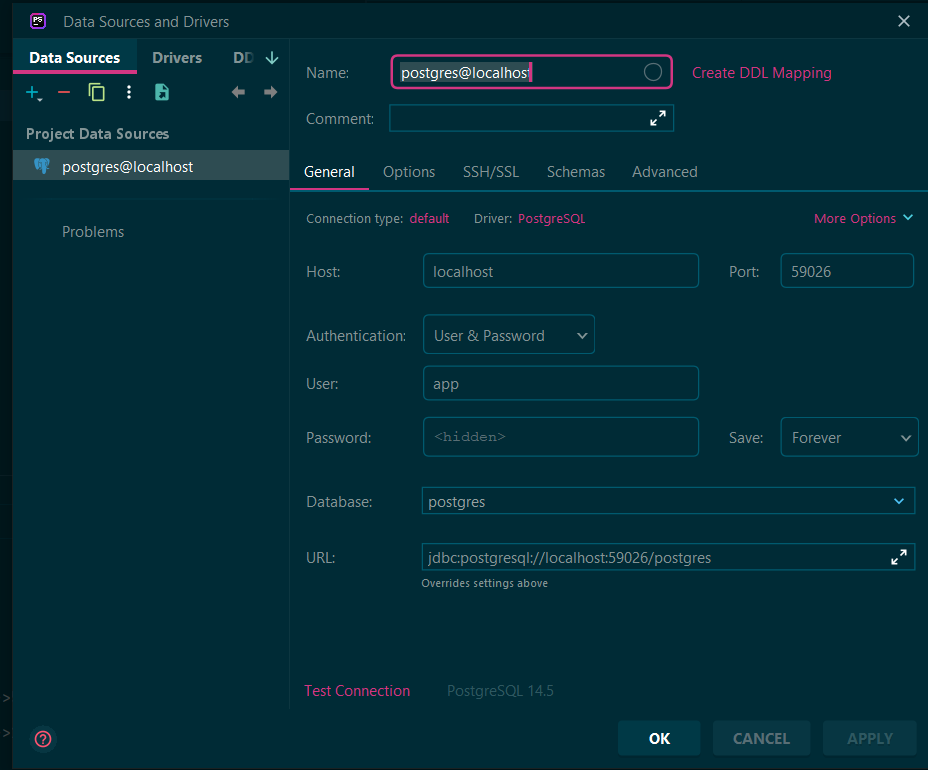
Attraverso il comando ‘docker-compose up -d’ docker crea e avvia il container lasciandolo in esecuzione in background.

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 ‘docker-compose.override.yml’ il numero di porta che Docker dovrà mappare.

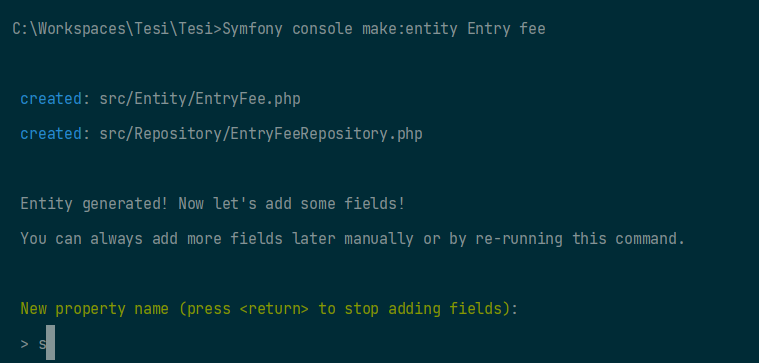
La porta da inserire sarà quella mappata da Docker (59026 nel nostro caso).

# Sviluppo della web app

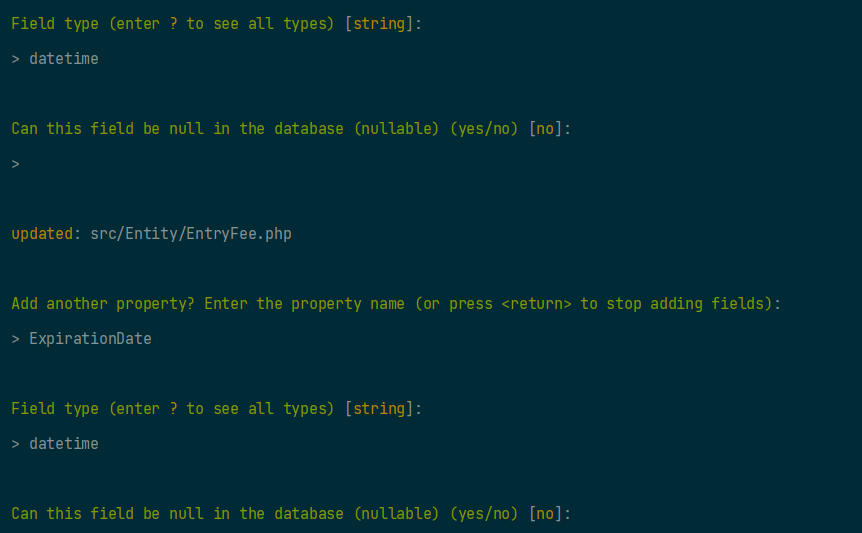
Si può procedere allo sviluppo vero e proprio del gestionale, per prima cosa quindi si procede alla creazione dello schema del database che conterrà le tabelle prima citate.

## 5.1 Creazione delle tabelle

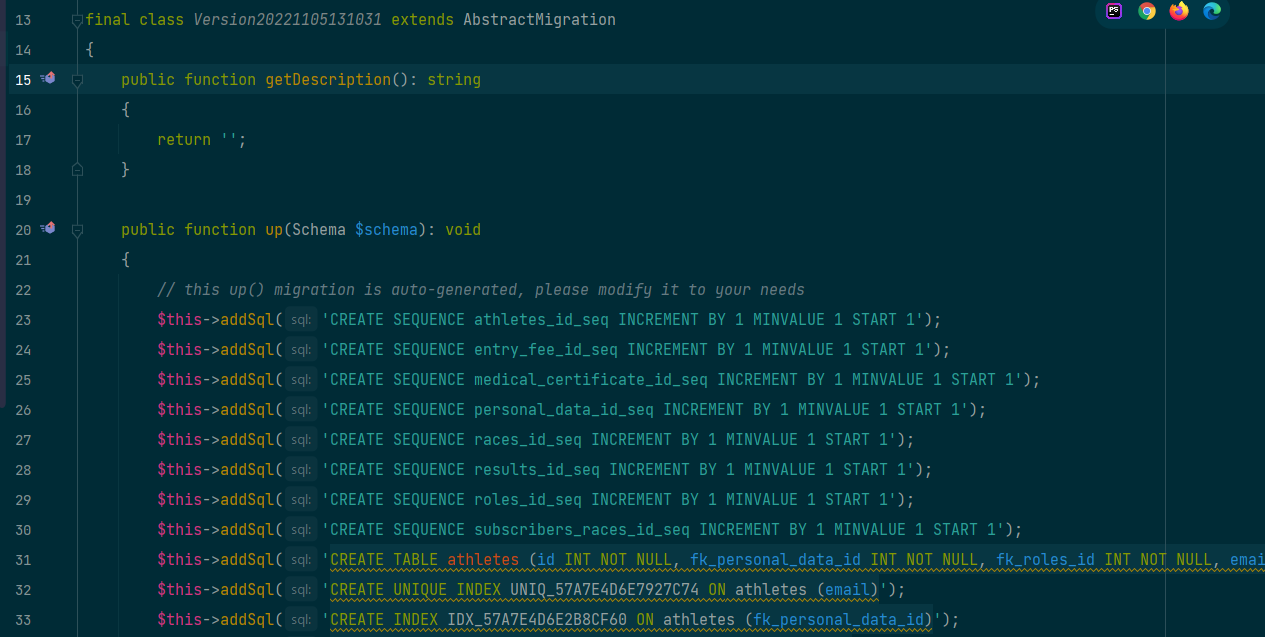
Grazie a Doctrine per creare l’entità non servirà scrivere i vari ‘create table’ ma basterà scrivere in console *Symfony console make:entity ‘NomeEntità’*



Ci verrà chiesto se aggiungere proprietà (attributi) e successivamente il loro tipo, ad esempio per l’entità Entry Fee:

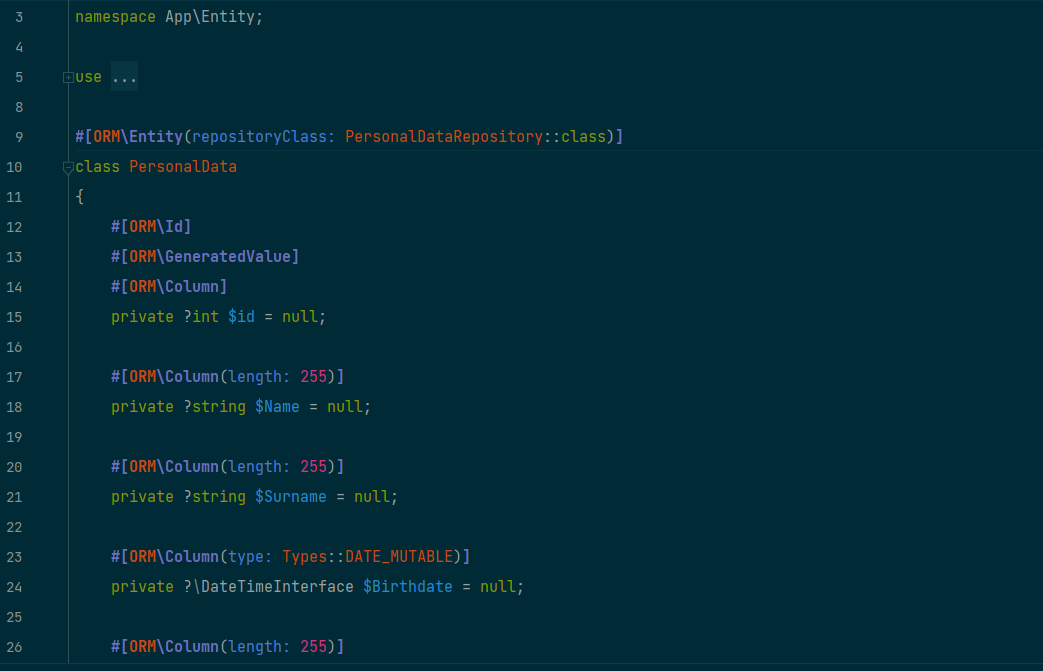


Alla fine del processo per tutte le entità l’ORM genera un file di migrazione in php che crea effettivamente le tabelle in linguaggio SQL:



## 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.



## 5.3 Creazione dei repository

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

# 10. Bibliografia e sitografia :

<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/>

<https://www.no-regime.com/ru-it/wiki/Running_boom_of_the_1970s>

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

<https://www.html.it/pag/16673/cos-php/>

<https://w3techs.com/>

<https://symfony.com/doc/current/best_practices.html#use-the-default-directory-structure>

<https://symfony.com/doc/current/quick_tour/the_architecture.html>

<https://symfony.com/legacy/doc/gentle-introduction/1_4/it/02-exploring-symfony-s-code>

<https://www.html.it/pag/390359/introduzione-a-symfony-5/>

<https://getbootstrap.com/>

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

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

<https://symfony.com/doc/current/components/dependency_injection.html>

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

<https://it.wikipedia.org/wiki/Unified_Modeling_Language>

Per i grafici sono stati utilizzati : (Yed graph, Visual paradigm Online)

Norme per la redazione dell’elaborato La relazione finale dovrà essere redatta dallo studente in modo standardizzato: - dimensioni: formato A4, ogni pagina di circa 35 righe, ciascuna di 65/70 caratteri di tipo Times New Roman, corpo 12, interlinea 1.5, margine di 3 cm per lato; - sul frontespizio devono essere indicati i dati principali (Università LUMSA, Dipartimento di Giurisprudenza, Economia, Politica e Lingue moderne, Corso di Laurea in…, titolo dell’elaborato in italiano e in inglese, nome e cognome dello studente, matricola, anno accademico, nominativo del docente referente); - l’indice e la bibliografia devono seguire le regole indicate dal docente referente. Per il frontespizio si veda il fac–simile pubblicato sul sito alla voce alla sezione Prova finale (https://www.lumsa.it/giurisprudenza-economia-politica-lingue-tesi).