# Progetto gestione di una palestra “MyGym”

Matteo Paterlini

L’applicativo è stato sviluppato in linguaggio python, utilizzando il framework MVC django.

## Traccia

A questo applicativo possono accedere diversi tipi di utente: utente anonimo, cliente, personal trainer e segretario.

Gli utenti anonimi possono navigare sulla home del sito e possono consultare e ricercare i corsi (con relativi prezzi) disponibili, ma per prenotarsi ad un corso devono registrarsi come clienti.

L’utente deve poter ricercare corsi per nome e prezzo massimo. Ogni corso ha una soglia massima di iscritti, dopodiché continuerà a comparire nei risultati di ricerca, ma non sarà più possibile iscriversi.

Una volta registrati al sito come clienti ci si può iscrivere ai singoli corsi (ogni cliente si può iscrivere una sola volta ad un corso), rinnovare ed allungare (o cancellare) eventuali iscrizioni già presenti.

L’utente potrà consultare presso la pagina personale dei singoli personal trainer alcune schede di allenamento caricate dagli stessi, e richiedere (selezionando una data ed un orario sul calendario del PT) una sessione di allenamento seguita. Questo prevede quindi che l'utente possa consultare l'elenco dei PT. Per il cliente deve inoltre essere presente una pagina profilo in cui consultare ed eventualmente modificare i propri corsi.

Gli utenti personal trainer possono visualizzare anch'essi ogni pagina che può visualizzare un cliente. Devono avere inoltre la possibilità di creare e modificare schede di allenamento, consultabili sul proprio profilo dai clienti.

Portinaio: può visualizzare l'elenco dei clienti e dei personal trainer, consultando i profili di entrambi. Sul profilo dell'utente deve poter visualizzare e modificare le iscrizioni di quest'ultimo, mentre sul profilo del personal trainer deve poter

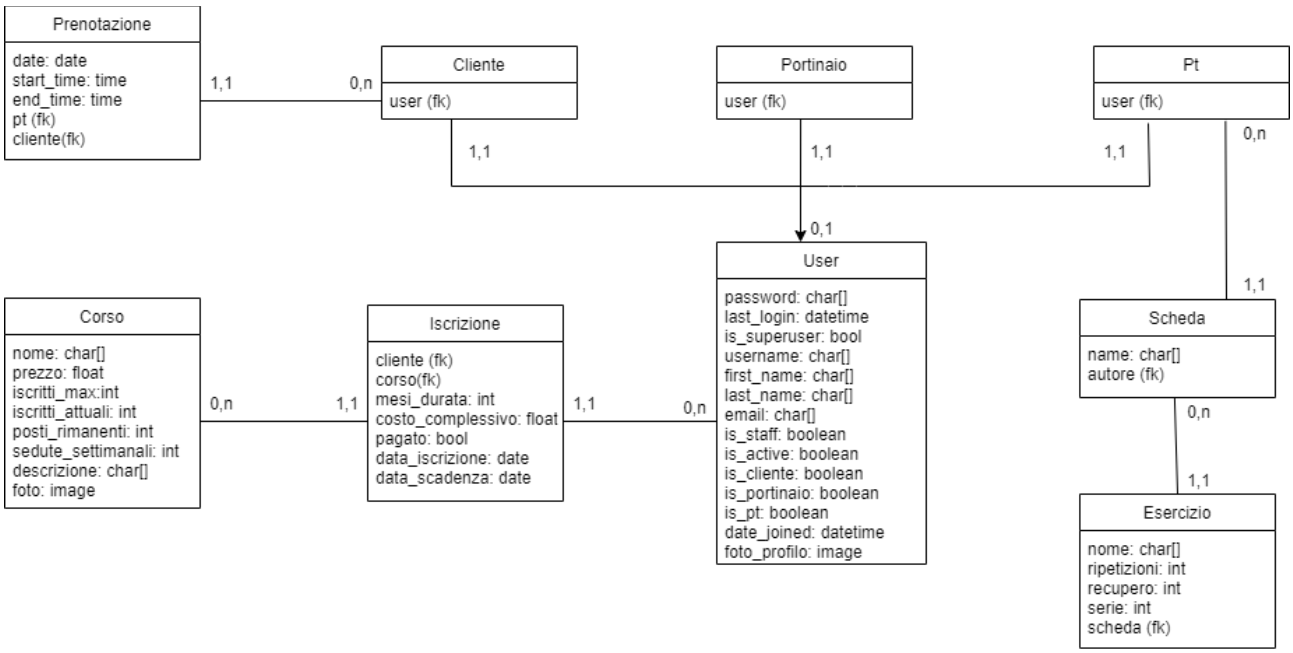
consultare il calendario di reperibilità. Può eliminare un corso dai corsi disponibili e può crearne di nuovi.

Ogni tipologia di utente deve potersi registrare con i suoi dati personali, una e-mail ed una foto profilo. Tale foto profilo deve poter essere aggiornata.

Non è presente la possibilità per l'utente anonimo di registrarsi come portinaio per motivi di sicurezza. Deve essere infatti l'amministratore tramite la piattaforma dedicata a creare un account per il portinaio. Tale piattaforma di amministrazione è fornita da django.

Siccome non è oggetto del corso gestire come avviene la transizione di denaro per un pagamento online, ho pensato di gestirlo nel seguente modo: nel momento in cui il cliente si iscrive ad un corso accumula un debito di cui viene tenuta traccia sul database. Tale debito può poi essere saldato dal portinaio una volta che l'utente si presenta fisicamente per saldarlo, andando sul profilo dell'utente e cliccando su un bottone che riporta a 0 questo debito (un paragone potrebbe essere fare acquisti su un sito di e-commerce e pagare alla consegna).

## Database




Il database che ho deciso di utilizzare per il progetto è stato di tipo Sql Lite, ovvero il database nativo di django.

La tabella **User** rappresenta raccoglie i dati di tutti gli utenti ed è utilizzata per l'autenticazione degli stessi. Contiene inoltre i dati necessari a determinare l'identità di un utente, dato vitale per limitare l'accesso ad alcune sezioni dell'applicativo.

Le tabelle **Cliente**, **Portinaio** e **Pt** contengono un riferimento alla tabella User e raccolgono gli identificativi dei singoli utenti che appartengono ad ognuna di queste categorie.

La tabella **Corso** contiene i dati relativi ad un corso a cui i clienti possono prenotarsi. Aggiorna in automatico i posti rimanenti per la partecipazione al corso.

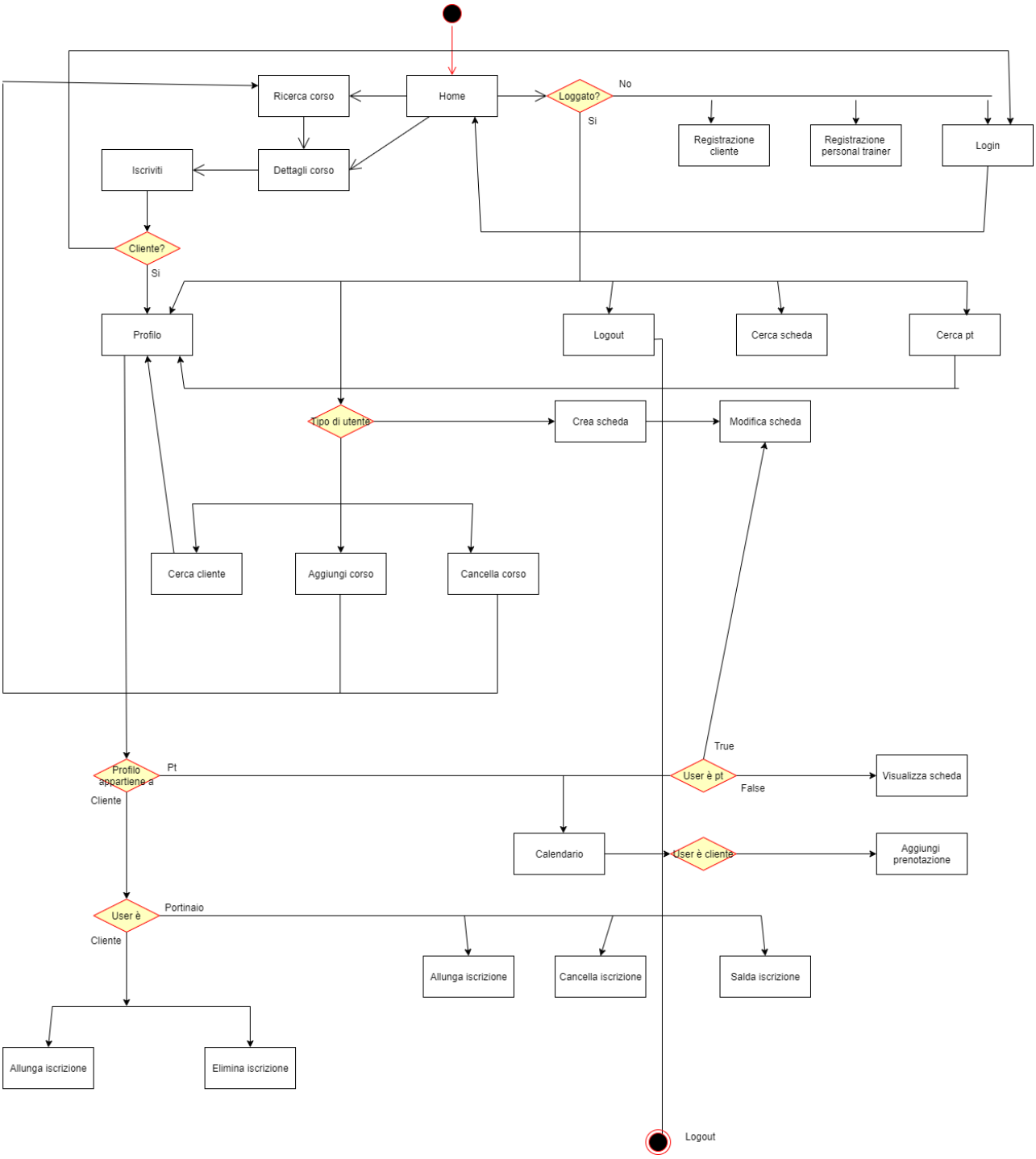
La tabella **Iscrizione** contiene invece i dati relativi ad una iscrizione da parte di un cliente ad un corso (incluse relative foreign key).

La tabella **Prenotazione** contiene le singole prenotazioni effettuate da un cliente che desidera una sessione di allenamento con un dato personal trainer sul calendario dello stesso.

La tabella **Scheda** è associata al personal trainer che la ha redatta, e serve a sua volta da riferimento ai singoli Esercizi.

La tabella **Esercizi** contiene i singoli esercizi che verranno visualizzati in una scheda di allenamento.

## Activity diagram




## Front-end

Per la realizzazione del front-end ho utilizzato HTML e CSS, facendo largo uso del framework Bootstrap. Le tabelle sfruttando django-tables2, mentre i form sono renderizzati grazie a crispy.

## Struttura progetto

Il progetto è suddiviso in 2 applicazioni: mygym e gym, ma il codice è sviluppato prevalentemente all'interno di gym. In questa applicazione possiamo infatti trovare la definizione dei modelli, la gestione della maggior parte degli url, i template che

verranno letti dal browser così come i template su cui si basano le tabelle ed i form (rispettivamente templates/gym, views.py e forms.py).

In gym/views.py sono definite le classi e le funzioni che gestiscono il funzionamento del sito.

In gym/utils.py è invece definito il comportamento e la struttura del calendario del personal trainer.

Infine, nel file gym/decorators.py vengono definiti dei decoratori personalizzati che regolano l'accesso alle funzioni presenti in gym/views.py sulla base del tipo di utente che richiede tale funzione.

In gym/models.py viene definita la classe User(AbstractUser), che verrà utilizzata per il login ed il signup (i quali vengono forniti da django).

```
class User(AbstractUser):
    is_cliente = models.BooleanField(default=False)
    is_pt = models.BooleanField(default=False)
    is_portinaio = models.BooleanField(default=False)
    foto_profilo = models.ImageField(upload_to='propic/', blank=True, null=True)

    @property
    def foto_profiloURL(self):
        try:
            url = self.foto_profilo.url
        except:
            url = 'images/placeholder.png'
        return url


class Cliente(models.Model):
    user = models.OneToOneField(User, on_delete=models.CASCADE, primary_key=True)

class Pt(models.Model):
    user = models.OneToOneField(User, on_delete=models.CASCADE, primary_key=True)

class Portinaio(models.Model):
    user = models.OneToOneField(User, on_delete=models.CASCADE, primary_key=True)
```

## Logica dell'applicativo

Come già detto, l'applicativo prevede l'accesso da parte di diversi tipi di utente: gli utenti anonimi, che possono navigare sulla home, ricerca corsi, consultarne i dettagli e registrarsi/iscriversi e gli utenti registrati. Nella home del progetto vengono visualizzati tutti i corsi a cui è possibile iscriversi (che hanno quindi dei posti disponibili).

Gli utenti loggati condividono la possibilità di visualizzare alcune pagine, ovvero la propria pagina di profilo, i profili dei personal trainer (e relativa pagina di ricerca), le schede di allenamento (e relativa pagina di ricerca), mentre altre pagine e funzionalità sono esclusive per ogni tipo di utente. Tutti gli utenti loggati hanno inoltre i privilegi di visualizzazione di un utente non loggato e possono aggiornare la propria immagine di profilo. Ogni utente deve avere una immagine profilo, che viene salvata nella cartella static/images/propic.

I clienti possono infatti prenotarsi ad un corso, cancellare o allungare tale iscrizione ed effettuare una prenotazione sul calendario di un personal trainer (che possono consultare).

I personal trainer possono consultare i vari calendari dei personal trainer, creare nuove schede di allenamento, e se una scheda è stata creata da un dato pt, questo pt può anche modificarla (aggiungendo o rimuovendo esercizi) o eliminarla. Possono, come i clienti, ricercare e visualizzare schede e personal trainer.

I portinai possono aggiungere o cancellare un corso, navigare tra i profili dei clienti e dei personal trainer, vedendone rispettivamente i corsi a cui sono iscritti e le schede, avendo anch'essi la possibilità di consultare i calendari dei personal trainer. Hanno inoltre il potere di saldare i debiti accumulati da un cliente iscritto ad un corso, e possono modificare o eliminare tale iscrizione.

Per limitare in modo efficiente l'accesso alle varie funzioni ho creato dei decoratori personalizzati, da utilizzare in aggiunta a @login_required.

```
#Controlla che l'utente sia un cliente
def cliente_required(function=None, redirect_field_name=REDIRECT_FIELD_NAME, login_url='gym:permesso-negato'):...

#Controllo che l'utente sia un pt
def pt_required(function=None, redirect_field_name=REDIRECT_FIELD_NAME, login_url='gym:permesso-negato'):...

#Controllo che l'utente sia un portinaio
def portinaio_required(function=None, redirect_field_name=REDIRECT_FIELD_NAME, login_url='gym:permesso-negato'):...
```

La navbar del sito deve cambiare dinamicamente in base al tipo di utente che lo sta consultando.

Oltre alle varie tipologie di utenti, un altro componente importante del progetto sono i corsi a cui un cliente si può iscrivere.

Ogni corso è caratterizzato da un nome, una foto (salvata in static/images), una descrizione ed un numero massimo di posti. In caso si raggiunga un numero di iscritti pari al numero massimo di posti disponibili non sarà più possibile ai clienti iscriversi a tale corso.

Il valore di posti disponibili viene aggiornato in automatico al momento del salvataggio

```
def save(self, *args, **kwargs):
    if self.posti_rimanenti is None:
        self.posti_rimanenti = self.iscritti_max
    else:
        self.posti_rimanenti = self.iscritti_max - self.iscritti_attuali
    super(Corso, self).save(*args, **kwargs)
```

e nel momento in cui raggiunge lo 0 la funzione non permette ad un nuovo cliente di iscriversi a tale corso.

```
@cliente_required()
def iscrizione_corso(request, id):
    corso = Corso.objects.get(id=id)
    if request.method == 'POST':
        if corso.posti_rimanenti > 0:
            form = IscrizioneAddForm(request.POST)
            if form.is_valid():
                iscrizione = form.save(commit=False)
                iscrizione.corso_id = corso.id
                iscrizione.cliente_id = request.user.id
                iscrizione.costo_complessivo = iscrizione.mesi_durata * corso.prezzo
                iscrizione.data_iscrizione = datetime.now()
                iscrizione.data_scadenza = datetime.now() + (iscrizione.mesi_durata*timedelta(days=30))
                iscrizione.save()
                corso.iscritti_attuali += 1 #Iscritti_rimanenti_si_aggiorna_da_solo
                corso.save()
                return redirect('gym:utente-details', id=iscrizione.cliente_id)
        else:
            redirect('gym:iscrizione-corso', id=id)
    else:
        form = IscrizioneAddForm
    logged_user = request.user
    context = {'logged_user': logged_user, 'form': form, 'corso': corso}
    return render(request, 'gym/iscrizione_corso.html', context)
```

L'iscrizione inoltre prevede un vincolo di unicità tra il cliente ed il corso a cui si è iscritto, prevenendo l'iscrizione multipla di un cliente allo stesso corso.

Le varie funzioni di ricerca hanno una struttura molto simile: ottengono un queryset dal database e lo filtrano sulla base dei dati inseriti dall'utente.

```
if form.cleaned_data['nome'] == form.cleaned_data['autore'] == '': #Campi_ricerca_vuoti
    schede = schede
else:
    schede = schede.filter(Q(nome__icontains=form.cleaned_data['nome'])&
                           Q(autore__username__icontains=form.cleaned_data['autore'])) #Filtro_ricerca
```

In questo caso, ad esempio, dopo aver inizializzato un queryset contenente i dati relativi a tutte le schede presenti sul database, effettuo un controllo sui campi del form di ricerca presentato all'utente: se tutti i campi sono vuoti, il queryset rimane invariato, altrimenti conterrà solamente gli oggetti che soddisfano tali requisiti.

# Unit testing

Nei miei test di unità ho testato la visualizzazione di una pagina (ricerca pt) da parte dei vari tipi di utente. In questo modo ho anche indirettamente testato la correttezza della funzione di login.

I test hanno tutti dato il risultato atteso, confermando il reindirizzamento dell'utente in caso non sia loggato, e renderizzando correttamente la pagina in caso l'utente sia loggato, indipendentemente dal tipo di utente.

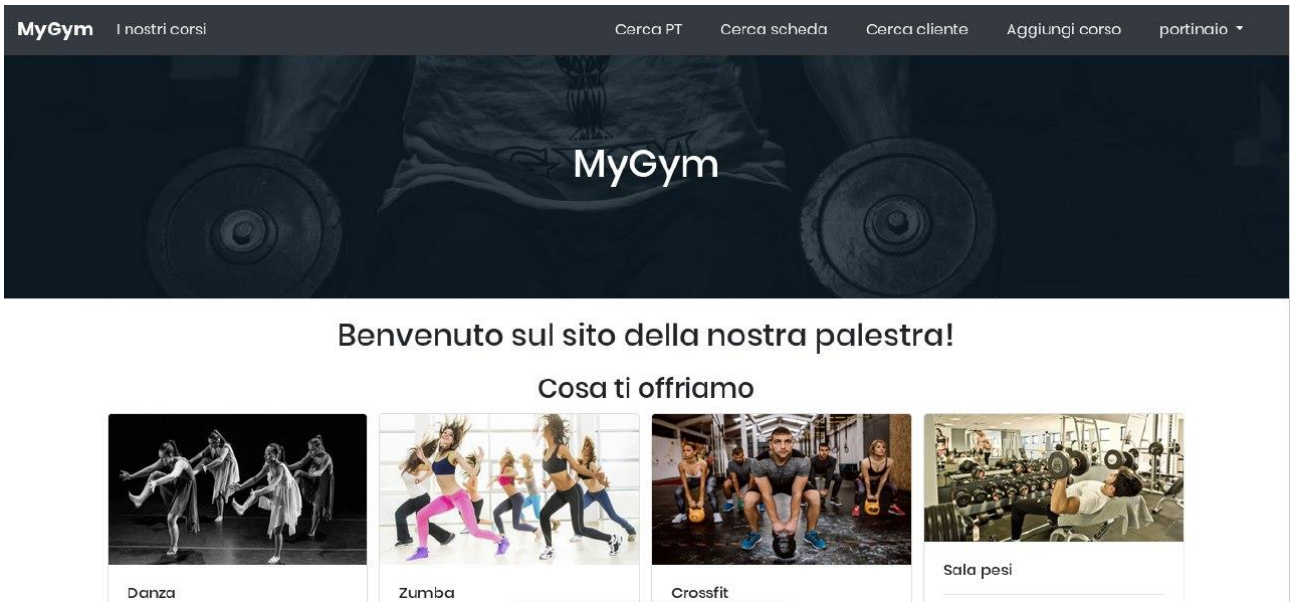
```
def test_profilo_no_logged_user(self):...

def test_profilo_logged_cliente(self):...

def test_profilo_logged_pt(self):...

def test_profilo_logged_portinaio(self):...
```

Ho inoltre testato il corretto funzionamento dell'aggiornamento automatico al momento del salvataggio del valore presente nel campo "posti disponibili" di un corso.

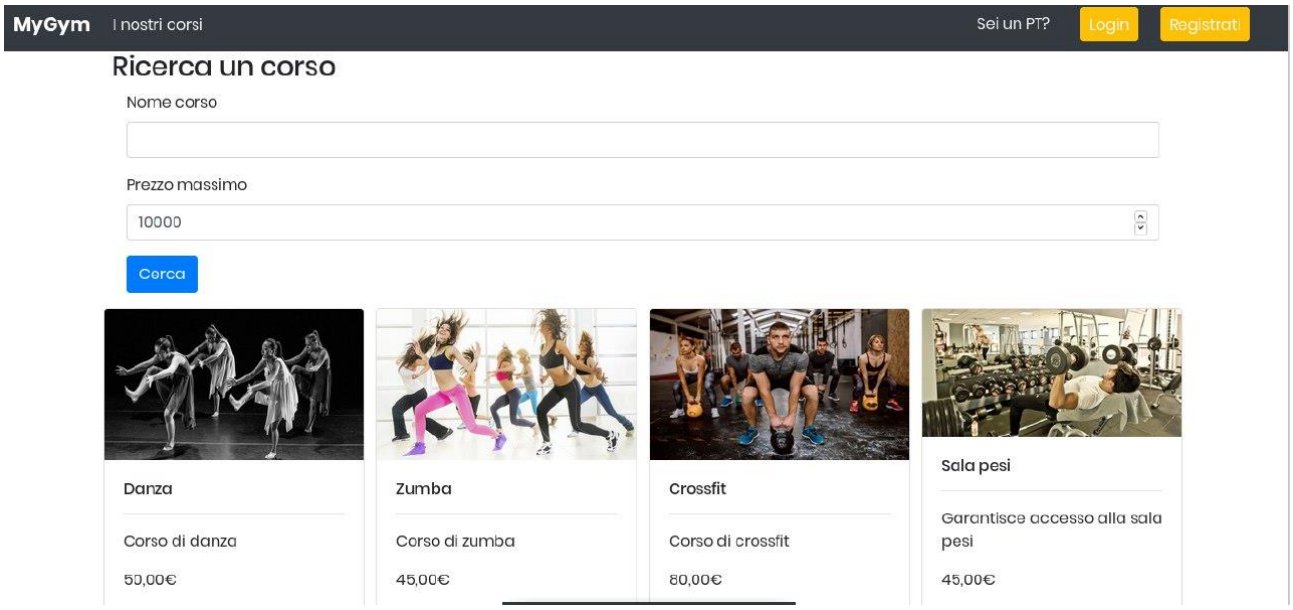
```
def test_posti_rimanenti(self):
    corso = Corso()
    corso.nome = "Corso"
    corso.prezzo = 5
    corso.iscritti_max = 10
    corso.sedute_settimanali = 3
    corso.descrizione = "Sono un corso"
    corso.foto = "zumba.jpg"
    corso.save()
    self.assertEquals(10, corso.posti_rimanenti)

    corso.iscritti_attuali = 3
    corso.save()

    self.assertEquals(7, corso.posti_rimanenti)
```

Screenshot del sito

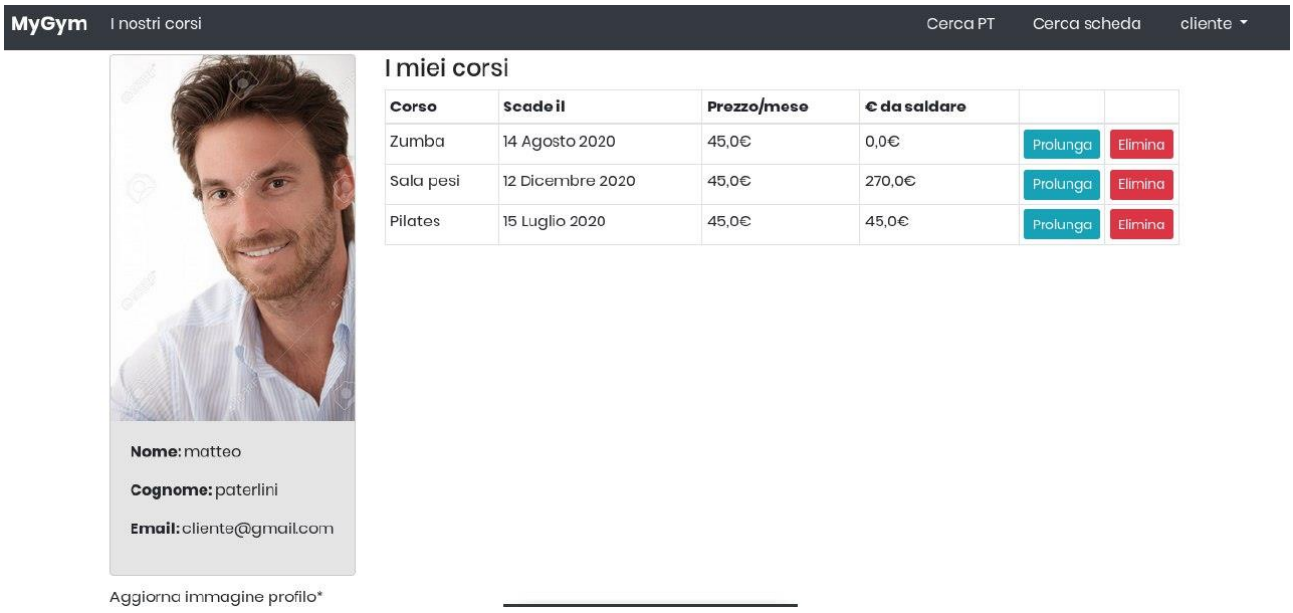
Home page




Ricerca corsi




Profilo cliente




# Profilo personal trainer