

Luca Lodesani (Mat. 165507)

# Tennis Coach

Tecnologie Web 11/07/2024

Luca Lodesani  
11/07/2024

# Introduzione

## Idea

Applicazione web per l'acquisto di **corsi online di tennis**, sia mentali che pratici. Essi possono essere a pagamento, o completamente gratuiti.

È rivolto a giocatori di tutti i livelli, grazie alla differenziazione dei corsi in diverse categorie. Una volta acquistati, gli utenti potranno visualizzare i propri corsi in una dashboard personale, dove saranno ordinabili in base a criteri specifici, come titolo, data d'acquisto, popolarità e prezzo. La suddivisione dei permessi tra le varie categorie di utenti ha permesso di garantire una **protezione** dei video.

Implementazione di un sistema di **pagamento** per consentire agli utenti di acquistare i corsi non gratuiti.

## Funzionalità

Più dettagliatamente, i requisiti funzionali sono i seguenti:

- Applicazione (essential) che permette di effettuare operazione CRUD sul DB, relativamente alle lezioni e ai corsi
- Applicazione (profile) che permette di gestire il profilo dell'utente, come la modifica/inserimento di nome, cognome, e-mail e descrizione. Sono implementate anche le funzioni di modifica dell'immagine profilo e cambio password
- Dashboard per la visualizzazione dei corsi acquistati/salvati dall'utente, con possibilità di ordinamento
- Dashboard dei corsi messi a disposizione da un coach, in modo da poterli visionare direttamente da un'unica pagina, con possibilità di ordinamento
- Pagine di errore (403-404) nel caso di accesso non autorizzato a risorse o a pagine non disponibili
- Sistema di messaggi in caso di operazioni andate a buon fine o in caso di errore, specialmente durante la fase di acquisto
- Form di ricerca dei corsi, con la possibilità di aggiungere filtri su
  - Campo di ricerca full-text search sul titolo
  - Coach che ha creato il corso
  - Livello (principiante, intermedio ed esperto)
  - Tipologia di costo (gratuito o a pagamento)
- Sistema di Login e Logout per consentire ai visitatori del sito web di avere le autorizzazioni necessarie per effettuare le operazioni, in base al loro ruolo

Sono presenti diverse tipologie di utente:

1. *Anonimo*: Accesso ai corsi per la sola visualizzazione, può controllare i dettagli di un corso senza avere la possibilità di acquistarlo. Per qualsiasi altra operazione è necessaria l'autenticazione; per questo motivo sono visibili i pulsanti di Log-in e Sign-up.
2. *Customer*: Utente base avente i permessi per interfacciarsi coi corsi e con le lezioni in maniera più approfondita. Esso può salvare i corsi gratuiti, o acquistare quelli a pagamento, in modo da aggiungerli alla propria dashboard. Ad ogni operazione è previsto un messaggio d'avviso. I video saranno accessibili solamente una volta acquistato il corso.
3. *Coach*: Utente in grado di poter aggiungere/modificare/cancellare i corsi, tramite una comoda pagina di gestione in cui potranno visionare e ordinare tutti i corsi da loro creati. Nella modifica di un corso sono comprese anche le operazioni di inserimento/modifica/cancellazione delle lezioni. Possiedono i permessi degli utenti precedentemente elencati, quindi potranno acquistare corsi messi a disposizione da altri coach.
4. *Admin*: Utente amministratore del sito web, avente i privilegi massimi; di conseguenza potrà accedere alle stesse pagine dei Coach. Tramite il pannello di controllo, può gestire le utenze e creare nuovi utenti "Coach", grazie a un pulsante messo a situato nel footer di ogni pagina del sito.

Sono presenti unit testing utili per verificare la correttezza della vista di salvataggio dei corsi e della logica per la gestione delle autorizzazioni di accesso ai video

Il sito è popolato automaticamente con degli utenti di test (4 utenti e 4 coach) e con dei corsi (~50) ognuno avente un numero variabile di lezioni (~2-8), in modo tale da poterne testare le funzionalità. I corsi sono stati presi da un file JSON custom, presente in tenniscoach/setup

## Permessi

Nella tabella riportata sotto sono rappresentate in modo schematico le autorizzazioni concesse ad ogni categoria di utente, per ciascuna tabella del database.

User	Course/Lesson	Purchase/Payment	Profile
<i>Anonymous</i>	View	Ø	Create
<i>Customer</i>	View	View, Create	View, Update <sup>3</sup>
<i>Coach</i>	View, Create <sup>1</sup> , Update <sup>1</sup> , Delete <sup>1</sup>	View, Create <sup>2</sup>	View, Update <sup>3</sup>
<i>Admin</i>	View, Create, Update, Delete	View, Create, Update, Delete	View, Create, Update, Delete

<sup>1</sup>: solo relativamente ai propri corsi/lezioni

<sup>2</sup>: solo per i corsi creati da altri utenti

<sup>3</sup>: solo per il proprio profilo

# Database

## Diagramma E-R

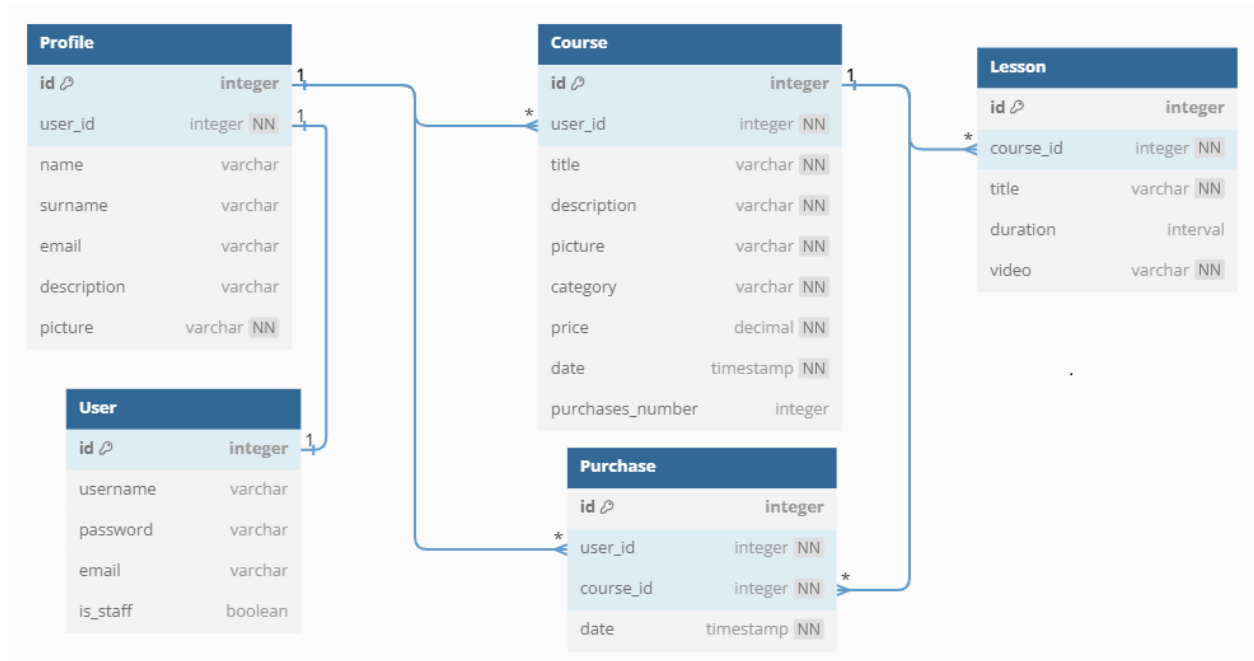


Figura 1

Osservando il diagramma ER in [Figura 1](#), sono presenti 5 entità nel complesso, quali *Course*, *Lesson*, *User*, *Profile* e *Purchase*.

Un *Course* è modellato con gli attributi mostrati in figura, ovvero titolo, descrizione, categoria (Principiante, Intermedio ed Esperto) ed un prezzo, eventualmente 0 se il corso è gratuito. La data non deve essere inserita dall'utente, e corrisponde alla data di creazione del corso. Inoltre, ogni corso dispone di una immagine che lo caratterizza, insieme al numero di volte che è stato acquistato.

Ogni *Course* è formato da un insieme di *Lesson*, ovvero videolezioni costituite da titolo e file video. La durata della lezione viene calcolata runtime, dopo aver inserito il file. Dal momento che una lezione si può trovare all'interno di un solo corso è presente la FK che punta direttamente a quello corrispondente.

Grazie all'entità *Profile* è stato possibile aggiungere alcuni dettagli sugli utenti come nome, cognome, immagine profilo, e-mail e descrizione. Questi ultimi due campi risultano utili particolarmente per i Coach. Questa entità è legata da una relazione uno-a-uno con *User*, la classe che modella un'utenza, fornita di base da Django.

L'acquisto è modellato dall'entità *Purchase*, permettendo così di tenere traccia dei corsi comprati dagli utenti. Gli attributi sono, di conseguenza, l'ID del corso soggetto all'acquisto dell'utente, anch'esso salvato nella tabella tramite il suo ID. Di fatto si tratta di FK "linkate" alle rispettive entità. Viene anche salvata una data, che fa riferimento al momento dell'acquisto.

## Specifiche

SQLite è il DBMS scelto per questo progetto, principalmente per motivi di convenienza, grazie alla maggior portabilità e semplicità di configurazione rispetto ai rivali PostgreSQL e MySQL.

Nello specifico si è voluto cercare di implementare una full-text search sui titoli dei corsi, funzionalità non supportata nativamente da SQLite. È stata resa fattibile ricostruendo la query, inserita nella searchbar della pagina di ricerca dei corsi, mettendo in AND ogni termine presente.

```
query = self.request.GET.get('query', '')
single_terms = query.split()
filter_gen = (Q(title__icontains=word) for word in single_terms)
query_fulltext = reduce(operator.and_, filter_gen, Q())
queryset = queryset.filter(query_fulltext)
```

Una volta estratti i singoli termini della query, viene creato un generatore di oggetti Q per ogni termine. Q è una classe Django utilizzata per costruire query complesse con condizioni logiche (AND in questo caso). Viene così costruita una query che cerca nel campo *title* del corso, insensitive, ogni parola presente in *single\_terms*. Tramite la *reduce* vengono combinati tutti gli oggetti Q generati in *filter\_gen* in una condizione singola composta, usata nell'istruzione dopo come filtro sul *queryset*.

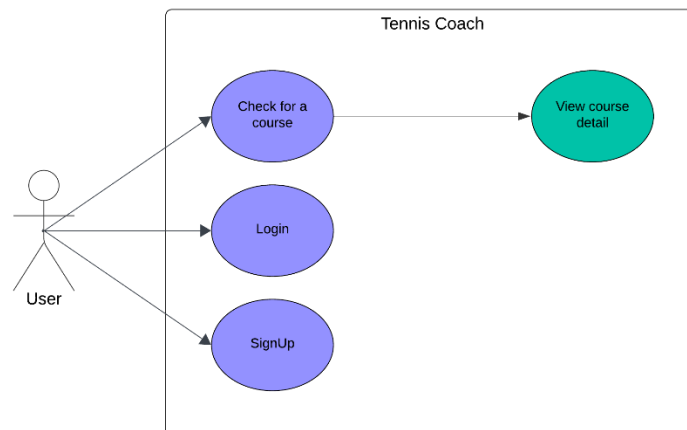
## Implementazione

La relazione tra *Course* e *Profile* può essere vista come molti-a-molti, dal momento che un corso può essere acquistato da molti utenti e un utente può acquistare diversi corsi. Di conseguenza la tabella *Purchase* può essere vista come una reificazione, contenente le *ForeignKey* che fanno riferimento alle entità *Profile* e *Course*, oltre ad una data. È quindi stato reso possibile linkare queste due entità grazie al campo *ManyToManyField* presente in *Profile*.

```
class Purchase(models.Model):
    user = models.ForeignKey(Profile, on_delete=models.CASCADE)
    course = models.ForeignKey(Course, on_delete=models.CASCADE)
    date = models.DateTimeField(auto_now_add=True)
    ...

class Profile(models.Model):
    user = models.OneToOneField(User, on_delete=models.CASCADE)
    name = models.CharField(max_length=20, null=True)
    surname = models.CharField(max_length=20, null=True)
    email = models.EmailField(null=True)
    description = models.CharField(max_length=400, null=True, blank=True)
    picture = models.ImageField(...)
    purchases = models.ManyToManyField(to = "essential.Course",
                                     through = "essential.Purchase",
                                     blank=True,
                                     related_name="profile_purchases")
    ...
```

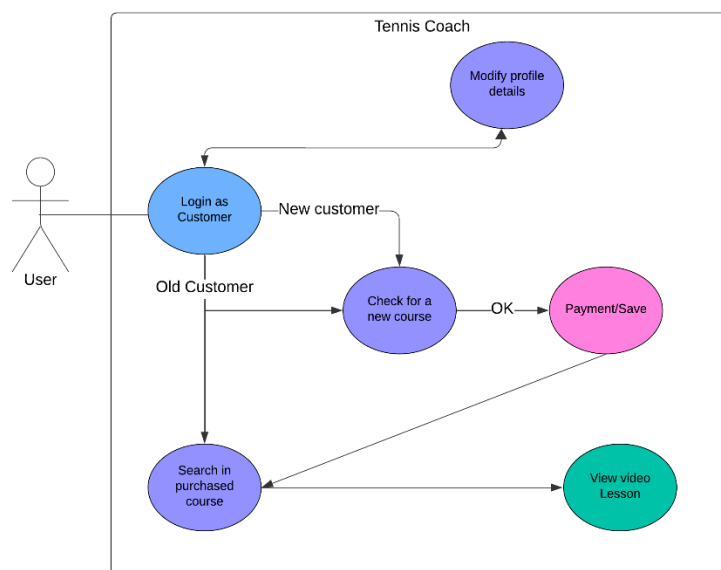
## UML: use case Anonymous



L'immagine rappresenta un diagramma UML di un caso d'uso per un utente anonimo.

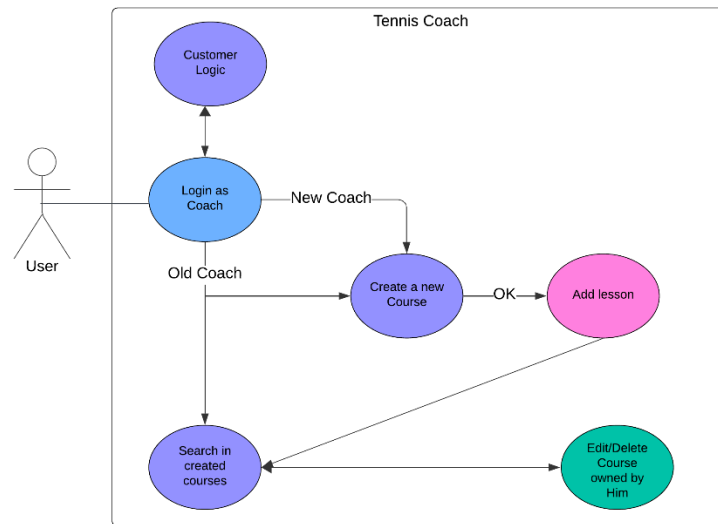
Un utente anonimo può navigare nel sito e controllare la lista dei corsi disponibili. Se trova un corso di interesse, può visualizzarne i dettagli, senza però avere le autorizzazioni per procedere con l'acquisto per vederne i video. Alternativamente, l'utente può decidere di effettuare il login, come *Customer* o *Coach*, se ha già un account, oppure iscriversi (sign-up) come *Customer*, se non è ancora registrato.

## UML: use case Customer



L'immagine rappresenta un diagramma UML di un caso d'uso per un *Customer*. Dopo il login, lo *User* diventato *Customer*, può seguire percorsi diversi a seconda che sia un nuovo cliente o un cliente già esistente. Un nuovo cliente può controllare i corsi, grazie al form di ricerca, e, se interessato, procedere al pagamento. Successivamente, può visualizzare le lezioni video del corso acquistato. Un cliente già esistente, invece, può anche cercare direttamente nei corsi precedentemente acquistati per poi accedere alle videolezioni.

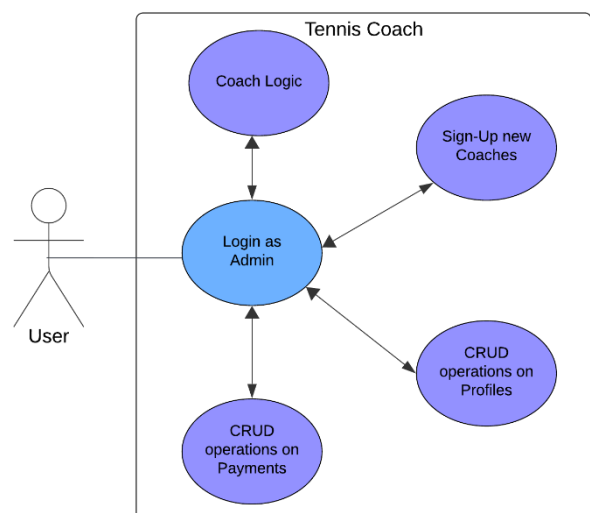
## UML: use case Coach



L'immagine rappresenta un diagramma UML dei casi d'uso per un coach di tennis. L'attore principale, l'utente, interagisce con il sistema attraverso varie funzionalità. Dopo essersi autenticato come *Coach*, può accedere alla “logica Customer”, ovvero eseguire tutte le operazioni elencate nel UML precedente. I coach, sia “nuovi” che “vecchi”, possono creare nuovi corsi. Dopo aver creato un corso, è possibile aggiungere lezioni ad esso, oltre a modificare/eliminare i corsi di cui sono proprietari. I coach hanno la possibilità di cercare i corsi da aggiornare direttamente in una dashboard dedicata.

## UML: use case Admin

L'immagine rappresenta un diagramma UML dei casi d'uso per un amministratore di un sistema di TennisCoach. In questo caso, una volta autenticato, l'admin può accedere alla “logica Coach”. Di conseguenza potrà accedere al sito proprio come se fosse un *Coach*, con l'aggiunta di poter registrare anche nuovi coach. Inoltre, l'amministratore può eseguire operazioni CRUD sui profili e sui pagamenti, ovvero creare, leggere, aggiornare ed eliminare le informazioni, in base a considerazioni personali. Queste funzioni permettono una gestione completa e centralizzata del sistema.



# Tecnologie

## Frontend

Per lo sviluppo del frontend, sono stati usati **HTML**, **CSS** e **JavaScript** (JS) come tecnologie di base. HTML (HyperText Markup Language) permette di definire la struttura delle pagine web, CSS (Cascading Style Sheets) per gestire l'aspetto visivo e il layout, mentre JavaScript (JS) aggiunge interattività e dinamismo.

Grazie a JavaScript è stato possibile implementare modali o messaggi che notificano all'utente l'avvenuto successo di login o di operazioni sul profilo e sui corsi. Per integrare queste notifiche è stata utilizzata **Toastr**, una libreria JS pensata per mostrare notifiche non bloccanti. Essa ha reso il sistema di avvisi più elegante e moderno.

Queste tecnologie sono integrate con **Bootstrap 5.3**, un framework che fornisce componenti predefiniti, come il carosello della home o le card dei corsi, per creare interfacce utente moderne. Inoltre, grazie all'utilizzo della libreria **jQuery** è stata semplificata la navigazione e la manipolazione del DOM (Document Object Model).

La quasi totalità delle interazioni tra client e server avviene mediante richiesta-risposta HTTP. Per gestire richieste client-server, in maniera asincrona, è stato usato **AJAX** (Asynchronous JavaScript and XML). Questo migliora l'esperienza utente, rendendo le applicazioni più fluide, veloci e reattive, senza dover ricaricare l'intera pagina ad ogni risposta del server. In particolare, in questo progetto, le risposte dal server al client sono in formato JSON. Nello specifico è stato utilizzato per implementare una funzionalità di suggerimenti dinamici mentre l'utente digita caratteri nel campo di input per selezionare un coach. Ad ogni inserimento, viene inviata una richiesta AJAX, di tipo GET, all'**API RESTful** endpoint, che risponde con i coach che hanno nel loro username quei caratteri.

## Backend

Dal lato del backend, tra le tecnologie utilizzate ci sono **Django Framework**, **Django signals** e **payments**.

Quest'ultima libreria di Django permette di semplificare l'integrazione dei pagamenti nei progetti Django, supportando diversi provider. Una volta installata (`django-payments`) e aggiunta alle `INSTALLED_APPS` è necessario specificare il provider di pagamento, in questo caso Stripe, in `settings.py` (vd. sezione [Pagamenti](#)).

Le Django Views, offerte dal Django Framework, compongono il backend. Queste viste elaborate lato-server compongono gli endpoint e forniscono la logica con cui vengono gestite le richieste HTTP. In seguito, forniscono risposte in formato HTML tramite il Django Template Engine. Per l'autenticazione degli utenti (login e logout) sono state usate le logiche messe a disposizione da Django fornite nel modulo `django.contrib.auth` e sono stati solo creati i template con `crispy-forms`.

Infine, i Django signals sono stati utilizzati per catturare eventi, o trigger, riguardanti ai database, consentendo di eseguire determinate azioni in modo automatico in risposta ad essi.



Quest'ultimi, dettagliatamente, nel progetto sono stati collegati come segue:

- al modello *User*

```
@receiver(post_save, sender=User)
def create_profile(sender, instance, created, *args, **kwargs):
    if created:
        Profile.objects.create(user=instance)
```

Tramite il segnale *post\_save* quando un nuovo utente viene creato (“created=True”), viene creato anche un *Profile* associato a questo utente. Così facendo viene automatizzato il processo di creazione dei Profili.

- al modello *Purchase*

```
@receiver(post_save, sender=Purchase)
def increment_purchases_number(sender, instance, created, **kwargs):
    if created:
        course = instance.course
        course.purchases_number += 1
        course.save()
```

In questo caso quando viene creato un nuovo acquisto ed inserito nella tabella *Purchase* viene aumentato il contatore *purchases\_number* del corso associato all'acquisto. Permette di automatizzare l'incremento del contatore sul numero di acquisti per ogni corso.

```
@receiver(post_delete, sender=Purchase)
def decrement_purchases_number(sender, instance, **kwargs):
    course = instance.course
    course.purchases_number -= 1
    course.save()
```

A differenza degli altri, qua il segnale è *post\_delete*. In tal modo, quando un acquisto viene eliminato (possibile solo dall'admin), in automatico, si decrementa il contatore *purchases\_number* del corso associato all'acquisto.

Grazie al framework **Django messages** è stato possibile gestire i messaggi, mostrati con la libreria *Toastr*. I “flash messages” permettono di comunicare in modo istantaneo informazioni, in modo tale da garantire che l'utente sappia sempre cosa sia successo al seguito di un click. L'invio di questi avvisi è stato predisposto ad esempio dopo:

1. aggiunta/modifica/inserimento di un corso e lezione
2. acquisto di un corso
3. modifica del profilo personale

# Organizzazione del codice

Il progetto con nome *tenniscoach* è suddiviso in tre applicazioni:

- **essential:** si concentra principalmente sulla gestione dei corsi e delle lezioni, fornendo funzionalità per crearli, mostrarli, modificarli ed eliminarli. Contiene i modelli *Purchase*, *Course* e *Lesson*, coi quali interagiscono numerose viste. Tra quelle più degne di nota troviamo:
  - *CoursesListView*: Mostra una lista di corsi disponibili impaginati e con form di ricerca/ordinamento
  - *CourseDetailView*: Mostra i dettagli di un corso specifico e le lezioni associate
  - *CreatedCoursesListView*: Mostra la lista dei corsi creati da un utente “Coach”, per permetterne la modifica in una pagina dedicata
  - *save\_course*: Permette agli utenti di salvare un corso se non ne sono i creatori e se hanno effettuato il pagamento, quando necessario.
  - *ajax\_search\_coaches*: Si tratta di una view che, inviando risposte JSON, permette di cercare i coach runtime con richieste AJAX

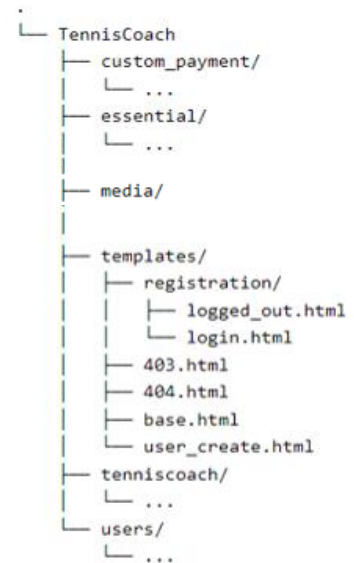
Oltre a queste sono presenti view accessibili solo da utenti “Coach”, ovvero quelle che permettono di fare operazioni di Update, Delete e Create sui corsi e lezioni

- **custom\_payment:** contiene il modello e la vista per gestire il pagamento dei corsi (vd. sezione [Pagamenti](#))
- **users:** contiene il modello *Profile* e fornisce la gestione completa dei profili utente, inclusa la loro visualizzazione, modifica e gestione della password, oltre a funzionalità specifiche per i Coach.

Oltre ai template necessari, contiene le seguenti viste:

- *ProfileDetailView*: Permette agli utenti di visualizzare il proprio profilo
- *modifica\_profilo*: Permette agli utenti di aggiornare il proprio profilo, mostrando un messaggio di successo.
- *CoachProfileDetailView*: Permette di visualizzare i profili dei coach, includendo nella paginazione i corsi associati ai coach.
- *YourCoursesListView*: Mostra un elenco dei corsi acquistati da un utente, con la possibilità di ordinarli
- *ChangePasswordView*: Gestisce il cambiamento della password per l'utente loggato

Nella top-level directory *templates* si trovano il template *base.html*, sul quale sono stati costruiti tutti gli altri template del progetto, i template per la gestione degli errori *403.html* e *404.html* e quelli per il login, logout (inseriti nella cartella *registration* come richiesto dall'app *auth*) e creazione dell'utente.



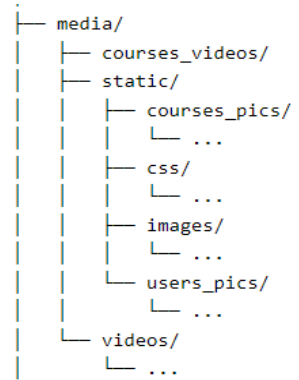
## Gestione dei media

Nel contesto di un'applicazione di gestione di corsi online come quella descritta, i file media (come i video delle lezioni) sono fondamentali per la UX, in quanto forniscono il contenuto cuore della piattaforma. Una buona gestione di essi risulta di cruciale importanza.

Si è cercato di renderla sicura ed appropriata, proteggendo i file video attraverso una gerarchia fissa come quella che si può vedere a destra.

I file all'interno di `media` sono divisi nelle directories nel seguente modo:

- `courses_videos`: video caricati nel sito a tempo di esecuzione
- `static`: contiene tutte le immagini e stili; non sono state protetti in quanto visibili da qualunque tipologia di utente.
  - `courses_pics`: immagini copertina dei corsi caricate tramite l'apposito form
  - `css`: contenente il foglio di stile CSS usato dalle pagine HTML
  - `images`: immagini di default usate dal file che inizializza il DB con dati presenti nel file json
  - `users_pics`: immagini di profilo caricate dagli utenti tramite il form del sito
- `videos`: video di default con cui vengono inizializzate le lezioni, comunque visibili solo dopo aver acquistato il corso corrispondente



Vengono instradate tutte le richieste per i file protetti alla funzione `authorize_media_resource` grazie a questa regola URL:

```
#Block unauthorized access to media
re_path(r'^media/.*$', authorize_media_resource),
```

La view personalizzata citata sopra è la seguente:

```
def authorize_media_resource(request: HttpRequest) -> HttpResponse:
    resource = os.path.basename(request.path)
    if request.user.is_authenticated and
        request.user.profile.has_permission_auth(resource):
        document_root = settings.MEDIA_ROOT
        media_path = request.path.split("media")[1]
        return serve(request, media_path, document_root, False)
    return redirect("403")
```

Nella pratica quello che viene fatto è:

- Servire il file, tramite la funzione *serve*, se l'utente ha i permessi richiesti; quest'ultimi vengono controllati con il metodo *has\_permission\_auth* messo a disposizione dai *Profile*

OPPURE

- Redirigere l'utente alla schermata di errore 403, nel momento in cui fa richiesta di una risorsa senza possederne i permessi per visualizzarla

La funzione *has\_permission\_auth* restituisce un valore booleano e controlla i seguenti campi:

```
def has_permission_auth(self, resource:str):  
  
    from essential.models import Lesson  
  
    lesson = Lesson.objects.filter(video__icontains=resource)  
    if not lesson:  
        return False  
  
    if lesson.filter(course__pk__in=self.purchases.values('pk')):  
        return True  
  
    if lesson.filter(course__user_id=self.user.id):  
        return True  
  
    return False
```

Dopo aver effettuato un lazy import, per evitare problemi di import circolari, e aver controllato che effettivamente esista una lezione con tale video, restituisce:

1. TRUE → se l'utente ha acquistato il corso che contiene quella lezione
2. TRUE → se l'utente è il creatore del corso che contiene quella lezione
3. FALSE → altrimenti

## Pagamenti

L'app dei pagamenti *custom\_payment* ha consentito di gestire le transazioni finanziarie tra gli utenti e la piattaforma di corsi online. Nel dettaglio consente agli utenti di acquistare i corsi in modo sicuro, efficiente ed il più realistico possibile, utilizzando un gateway di pagamento affidabile come Stripe.

È integrato un processo di checkout user-friendly per migliorare la UX.

È stato deciso di affidarsi a Stripe in quanto fornisce carte di prova (4242 4242 4242 4242) per testare le integrazioni senza effettuare transazioni reali. Inoltre, è possibile accedere alla dashboard di Stripe, il che facilita il processo di sviluppo e monitoraggio

È stato esteso il modello *BasePayment* di *django-payments*, che mette a disposizione numerosi campi per la gestione degli acquisti. Tra tutti i campi sono stati sfruttati in realtà solamente:

- *billing\_first\_name*: in cui è stato salvato lo username dell'utente che ha fatto l'acquisto
- *description*: per salvare il corso che è stato acquistato dall'utente
- *total*: prezzo del corso acquistato

Mentre l'ultimo ha un solo scopo informativo, i primi due vengono controllati nel momento del salvataggio dell'acquisto in *Purchase*, in modo tale da permetterlo solo previo acquisto.

La vista (*checkout*) che gestisce il checkout del pagamento:

1. Controlla l'esistenza del corso, in caso contrario reindirige alla pagina d'errore "404"
2. Verifica che l'utente non abbia già acquistato il corso e che non ne sia il proprietario, reindirizzandolo in caso contrario alla pagina predisposta con un messaggio di notifica.
3. Crea l'oggetto *Payment* se il form è valido ed invia una richiesta a Stripe per creare un intento di pagamento *PaymentIntent*, necessario per gestire il pagamento
4. Se non si tratta di una richiesta POST, mostra il template di checkout con le relative informazioni

Nel file *settings.py* è stato necessario aggiungere alcune righe, in particolar modo per definire il provider di pagamento, le chiavi pubbliche e private personali, oltre all'indirizzo dell'host dove l'applicazione è in esecuzione ("localhost").

## Unit Testing

Le funzionalità coperte dallo unit testing sono:

- La funzione *has\_permission\_auth*, che come scritto sopra, racchiude la logica per verificare se un utente può accedere ad una risorsa video
- La view *save\_course*, che fornisce la logica per il salvataggio del corso; grazie a questa *TestClass* è stato possibile testare tutti i casi che possono verificarsi al salvataggio di una entry nella tabella *Purchase*

I test si trovano, rispettivamente, in *users/tests.py* e in *essential/tests.py*

# Interfaccia

Per consentire una navigabilità fluida nel sito, si è pensato di mantenere una **navbar** sempre visibile nella parte alta dello schermo, costituita da:

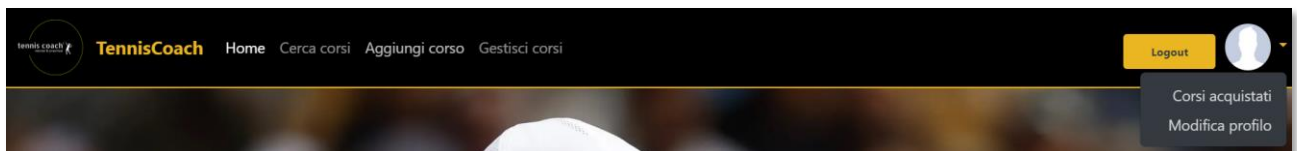
- **Logo e titolo** del progetto, cliccabili e rimandano alla home
- **Home**, al click rimanda alla home
- **Cerca corsi**, al click porta alla pagina contenente la lista dei corsi con i filtri di ricerca e ordinamento

Se ci si logga come *Coach* o *Admin* appaiono altre due sezioni:

- **Aggiungi corso**, che porta alla pagina di creazione del corso
- **Gestisci corsi**, che rimanda alla pagina coi corsi creati dall'utente loggato

Come si può vedere, nell'angolo in alto a destra della navbar è presente il tasto per effettuare il logout, se si è loggati, e l'immagine profilo dell'utente che al click fa scendere un menu a tendina.

Con esso sarà possibile accedere ai **Corsi acquistati** e alla pagina che per la **modifica del profilo**.



La navbar cambia se non si è loggati e mostra in alto a destra i pulsanti per effettuare l'accesso come *Coach* o *Customer*, od eventualmente per registrarsi come *Customer*.



Nella Home è presente un carosello che scorre in automatico e permette di accedere ad alcune pagine, in base alla categoria dell'utente che sta navigando il sito (loggato o anonimo)



In fondo ad ogni pagina è presente un footer che permette di accedere al pannello dell'Admin e, nel caso di accesso come Admin al sito, di iscrivere nuovi *Coach*



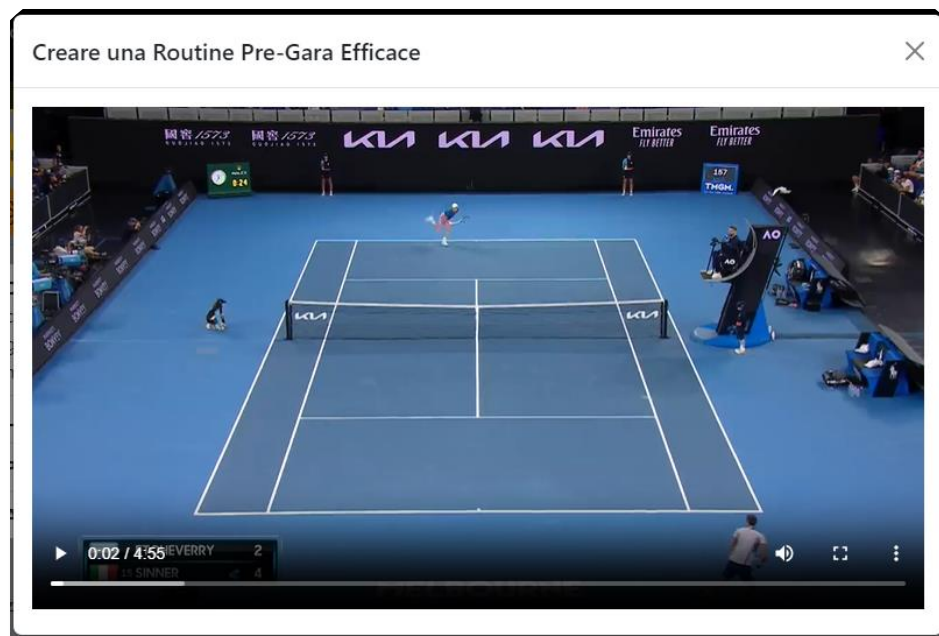
Footer da utente anonimo o loggato come Customer o Coach



Footer da utente loggato come Admin

La maggior parte degli elementi sarà sottoforma di **card**, e in caso fossero in numero elevato verrà fatta una paginazione, gestita dal paginatore di Django, che ritorna 6 o 9 elementi per pagina, in base alla pagina.


In seguito, alcuni risultati ottenuti:



Interfaccia per visualizzazione del video




☐ A pagamento☐ Gratis



Fondamenti del Tennis per Principianti

Principiante Acquistato: 0 volte


19,99€ Coach: ugolini



Preparazione Mentale per il Tennis

Principiante Acquistato: 0 volte


22,99€ Coach: mezzanotte



Strategie di Gioco per Principianti

Intermedio Acquistato: 0 volte


18,99€ Coach: ugolini



Gioco di Volée: Domina la Rete

Intermedio Acquistato: 0 volte


27,99€ Coach: menabue



Tennis per Bambini: Divertimento e Apprendim...

Esperto Acquistato: 0 volte

15,99€ Coach: mezzanotte




Tecniche di Ritorno del Servizio

Esperto Acquistato: 0 volte

26,99€ Coach: ugolini

44 risultati trovati | Pagina 1

Lista dei corsi con filtro di ricerca e ordinamento



**Andrea Prampolini**




Username: prampolini

Email: andrea@gmail.com

**Descrizione:** Sei alla ricerca di un coach di tennis esperto e appassionato che possa aiutarti a migliorare il tuo gioco e raggiungere i tuoi obiettivi tennistici? Non cercare oltre! Con oltre 20 anni di esperienza nel campo del tennis, Andrea è un allenatore certificato che ha aiutato numerosi giocatori a perfezionare le loro abilità, dai principianti agli agonisti.

**I suoi corsi...**

Ordina per:




Pagina dei corsi messi a disposizione da un coach



I tuoi corsi

Salva

Tecnica del Rovescio: Perfezionare il Colpo



**Descrizione:** Un corso dedicato a perfezionare la tecnica del rovescio, uno dei colpi più importanti nel tennis.

**Livello:** Intermedio

**Prezzo:** Gratis


**Acquistato:** 1 volta

Lezioni:

Fondamentali del Rovescio  
04m 55s

Rovescio a Una Mano vs Rovescio a Due Mani  
04m 55s



Esercizi di Pratica per il Rovescio  
04m 55s

Creato da: [prampolini](#) 


in data: Giovedì 04 Luglio 2024 11:56

*Corso visto da utente anonimo o loggato, ma senza averlo acquistato*

I tuoi corsi

Tecnica del Rovescio: Perfezionare il Colpo



**Descrizione:** Un corso dedicato a perfezionare la tecnica del rovescio, uno dei colpi più importanti nel tennis.

**Livello:** Intermedio

**Prezzo:** Gratis


**Acquistato:** 1 volta

Lezioni:

Fondamentali del Rovescio  
04m 55s

Rovescio a Una Mano vs Rovescio a Due Mani  
04m 55s

Esercizi di Pratica per il Rovescio  
04m 55s

Creato da: [prampolini](#) 

in data: Giovedì 04 Luglio 2024 11:56

*Corso visto dal proprietario del corso, o dall'Admin, con possibilità di modifica completa*