

# Athena Course Shop

Relazione di progetto realizzato per il corso di Tecnologie Web

Stefano Vezzalini

A.A.: 2020/2021

# Indice

<b>1</b>	<b>Il progetto</b>	<b>2</b>
1.1	I corsi . . . . .	2
1.2	Gli utenti . . . . .	2
1.3	Sezione recensioni e commenti . . . . .	3
1.4	Diagramma dei modelli UML . . . . .	4
<b>2</b>	<b>Tecnologie utilizzate</b>	<b>5</b>
2.1	Django . . . . .	5
2.2	Bootstrap . . . . .	6
2.3	Librerie utilizzate . . . . .	6
2.3.1	Django Crisy Forms . . . . .	6
2.3.2	Django Filters . . . . .	6
2.3.3	PyVimeo . . . . .	6
<b>3</b>	<b>Organizzazione logica dell'applicazione</b>	<b>7</b>
<b>4</b>	<b>Punti chiave e scelte fatte</b>	<b>8</b>
4.1	L'upload di nuovi corsi . . . . .	8
4.2	Cancellazione corsi . . . . .	8
4.3	Filtri per la ricerca . . . . .	9
4.4	I mixins . . . . .	9
<b>5</b>	<b>Test Realizzati</b>	<b>10</b>
5.1	Testing delle View . . . . .	10
5.1.1	Test di creazione di un nuovo corso . . . . .	11
<b>6</b>	<b>Risultati ottenuti</b>	<b>12</b>

# Capitolo 1

## Il progetto

Per il progetto ho voluto realizzare un sito web in cui gli utenti potessero pubblicare e vendere corsi di diversi tipi e categorie.

### 1.1 I corsi

Per i corsi ho scelto di appoggiarmi a Vimeo come servizio di streaming e alle sue API; ogni volta che un utente vuole caricare un nuovo corso il backend del sito si occuperà di caricare su Vimeo il file mp4 e di memorizzare l'id univoco del video che verrà poi utilizzato per effettuare l'embedding sulla pagina di visualizzazione.

Allo stesso modo quando un utente decide di cancellare il suo corso il backend gestirà anche l'eliminazione del file memorizzato su Vimeo tramite API.

### 1.2 Gli utenti

Al momento della registrazione ogni utente può decidere di iscriversi come **insegnante** o come semplice **studente**. Gli utenti di tipo insegnante sono dotati di alcune funzionalità aggiuntive all'interno del sito legate alla pubblicazione di corsi e alla visione di dati su questi ultimi. Un utente insegnante può comunque essere anche studente allo stesso tempo acquistando corsi di altri insegnanti partecipanti al sito. Gli utenti studente hanno la possibilità di acquistare corsi e successivamente di partecipare attivamente al sito tramite recensioni e commenti. Gli utenti anonimi (senza autenticazione) non possono acquistare o vedere i corsi ma possono comunque visualizzare i commenti e le recensioni di questi ultimi.

Ogni utente ha un suo profilo in cui può mostrare una breve descrizione di

se, visualizzare i corsi acquistati e monitorare i corsi pubblicati nel caso fosse un insegnante.

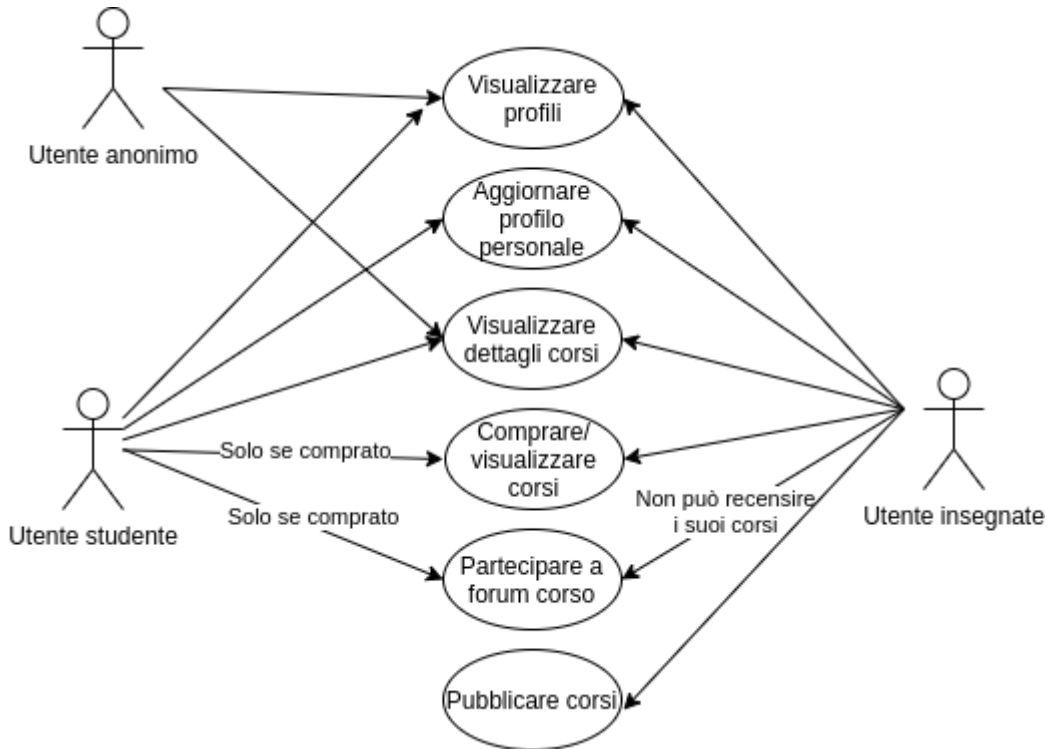


Figura 1.1: Use Case UML per gli utenti

### 1.3 Sezione recensioni e commenti

Ogni corso possiede la sua sezione commenti e la sua sezione recensioni, tutti gli utenti possono visualizzare queste sezioni ma solamente gli utenti che hanno acquistato o pubblicato il corso in questione possono parteciparvi.

La sezione commenti in particolare è stata pensata come uno spazio in cui utenti e insegnanti possano scambiarsi domande e opinioni sul corso acquistato ed è stata realizzata sulla falsa riga di quello che può essere un sito di video sharing come Youtube in cui ogni commento ha a sua volta una sezione risposte.

La sezione recensioni invece è accessibile solo dagli utenti che hanno acquistato il corso in modo da poter lasciare un voto e un commento con la possibilità, in futuro, di modificarli.

## 1.4 Diagramma dei modelli UML

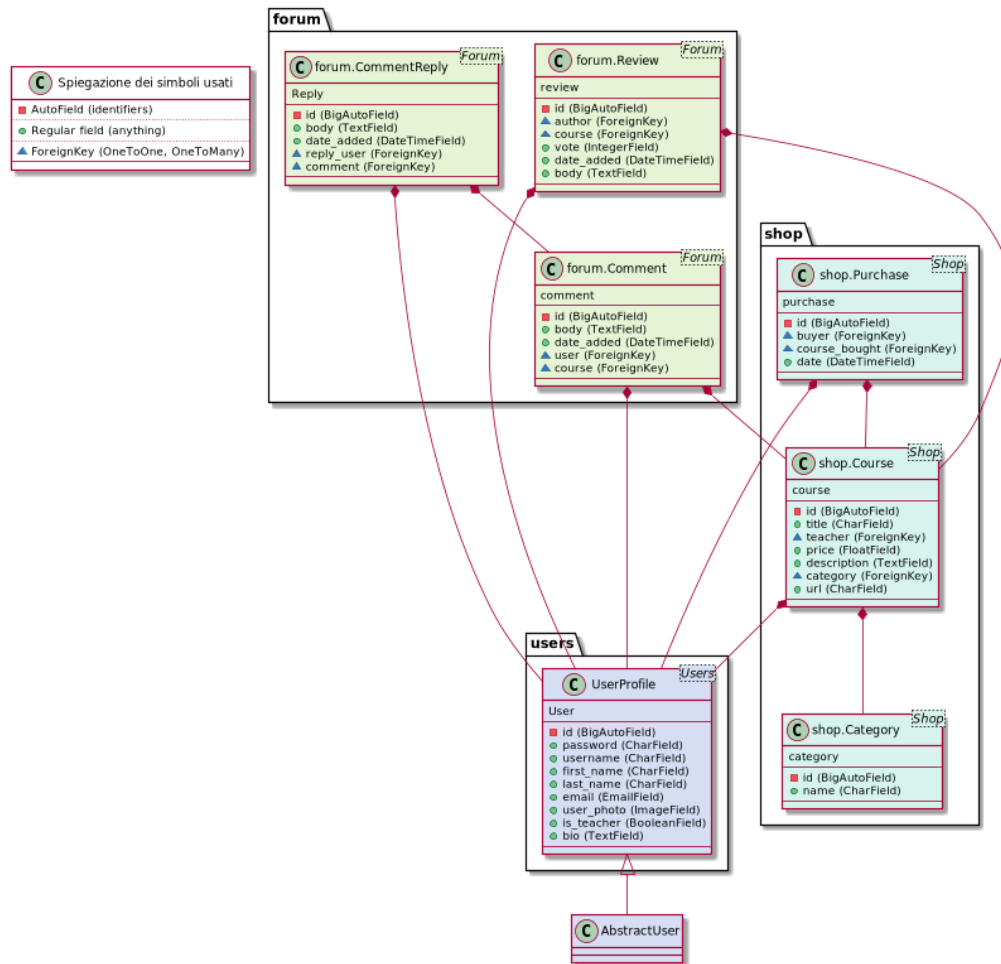


Figura 1.2: Diagramma UML dei modelli e delle loro relazioni

# Capitolo 2

## Tecnologie utilizzate

Per rendere più veloce e funzionale lo sviluppo del progetto ho deciso di utilizzare il web framework Django che fornisce di default molte delle funzionalità che ogni sito basato su database necessita al giorno d'oggi. A Django ho poi integrato due librerie: `django-crispy-forms` e `django-filters` che forniscono rispettivamente degli strumenti per creare form e per aggiungere strumenti di filtraggio all'interno di una pagina. Per la parte grafica ho scelto Bootstrap, una raccolta di componenti basati su CSS che permettono di default di ottenere un'applicazione web responsive su ogni tipo di dispositivo.

### 2.1 Django

Django è un web framework con licenza open-source per lo sviluppo di applicazioni web scritto in Python seguendo il modello "Model-Template-View". Django permette di rendere facile e veloce la creazione di applicazioni web basate su database.

Django segue il modello **Model Template View**:

- **Model**: Un model è l'astrazione di un oggetto che lo sviluppatore vuole rappresentare all'interno dell'applicazione e che verrà memorizzato nel database. Un model viene realizzato dal programmatore come una classe python a tutti gli effetti
- **Template**: i template sono l'elemento più vicino all'utente finale, forniscono l'interfaccia grafica dell'applicazione e permettono all'utente di fornire gli input
- **View**: permettono di collegare i models e i template, si occupano di richiedere i dati necessari al database e di gestire tutte le operazioni

di modifica, aggiornamento e creazione. Allo stesso tempo ricevono e gestiscono gli input inseriti dall'utente e "inviano" i dati ai template

Django supporta nativamente diversi database, per il progetto ho deciso di rimanere su quello di default ovvero SQLite.

## 2.2 Bootstrap

Bootstrap è un framework CSS open source realizzato da Twitter e volto ad uno sviluppo responsive e mobile-first.

Bootstrap fornisce di default una grande quantità di elementi e un tema di default per creare facilmente siti e applicazioni web visivamente gradevoli. È anche possibile trovare online gratuiti e a pagamento temi nel caso ci si volesse distaccare da quello di base.

## 2.3 Librerie utilizzate

### 2.3.1 Django Crispy Forms

Django Crispy Forms è una libreria che implementa diverse funzionalità che permettono di mostrare form più eleganti e consente uno sviluppo DRY (Don't Repeat Yourself) di questi ultimi.

### 2.3.2 Django Filters

Django Filters è una libreria che assiste l'utente nella scrittura di filtri per mostrare solo certi oggetti all'interno di una pagina.

### 2.3.3 PyVimeo

PyVimeo è la libreria ufficiale per Python di Vimeo, funge da wrapper per Requests permettendo un'interazione più facile e meno verbosa con le API del servizio.

## Capitolo 3

# Organizzazione logica dell'applicazione

L'applicazione è organizzata in quattro apps:

- **Shop:** l'app che contiene tutto quello che riguarda il negozio e i corsi in vendita
- **Users:** l'app che contiene il modello dell'utente e i template e le view che permettono di visualizzarne il profilo e di aggiornarlo
- **Authentication:** l'app che contiene tutte le componenti che gestiscono registrazione, login e logout
- **Forum:** l'app che contiene le componenti che riguardano commenti e recensioni

Per decidere quali app creare ho ragionato in termini di riusabilità di queste ultime oltre a voler creare una separazione logica dei diversi componenti. Ad esempio le app shop e forum inizialmente erano fuse insieme ma successivamente ho deciso di separarle pensando al fatto che le funzionalità che sono nell'app forum potrebbero essere utili e implementate in altre applicazioni web.

Per quanto riguarda l'app users e l'app authentication ho ritenuto più adeguato separare le funzionalità più ad "alto livello" che riguardano un utente come la visualizzazione del profilo dalle funzionalità riguardanti la sola autenticazione.



## Capitolo 4

# Punti chiave e scelte fatte

### 4.1 L'upload di nuovi corsi

La parte che ha richiesto più lavoro all'interno del progetto è stata quella della gestione dell'upload di nuovi corsi sul portale. Per la memorizzazione dei file video ho utilizzato Vimeo e le sue API. Per la view di upload (`CreateViewVimeo`) ho fatto l'override sia del metodo `post` che del metodo `get`, in particolare il metodo `post` riceve prima i dati e il file video inseriti dall'utente nel form, dopodichè verifica che siano validi ed esegue l'upload su Vimeo ricevendo un codice identificativo; quest'ultimo verrà utilizzato per inserire un nuovo campo nella tabella `Courses` nel database.

Il codice identificativo permette poi di inviare query a Vimeo per ottenere diverse informazioni sul video come ad esempio tre thumbnail di diverse dimensioni.

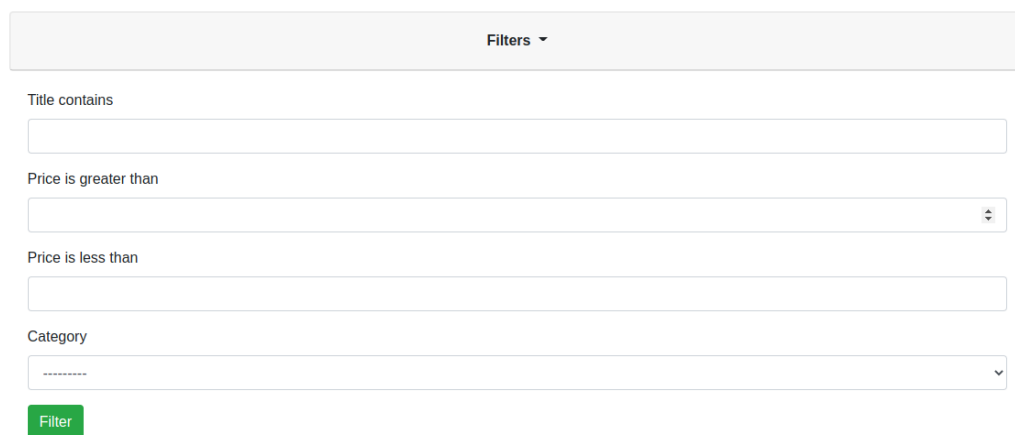
Per interfacciarsi con le API ho utilizzato `PyVimeo`, la libreria python ufficiale di Vimeo.

### 4.2 Cancellazione corsi

In modo simile all'upload di nuovi corsi per la cancellazione ho effettuato l'override del metodo `delete` all'interno della view apposita (`CourseDelete`) in modo tale che prima di rimuovere la riga dal database effettui una chiamata che elimini il file video anche sul mio account Vimeo

## 4.3 Filtri per la ricerca

Per implementare una serie di filtri sulla ricerca ho utilizzato la libreria `django-filters` all'interno della `ListView` che mostra tutti i prodotti in vendita. I filtri sono stati definiti nel file `filters.py` e permettono di dettagliare la ricerca attraverso parole chiave contenute nel titolo, cercando corsi che siano in un certo range di prezzo e cercando solo corsi appartenenti a una categoria in particolare.



The image shows a web form for filtering search results. At the top, there is a header bar with the text "Filters" and a downward arrow. Below this, the form is organized into several sections: "Title contains" with a text input field; "Price is greater than" with a text input field and a small upward arrow icon; "Price is less than" with a text input field; and "Category" with a dropdown menu showing "-----". At the bottom left of the form is a green button labeled "Filter".

Figura 4.1: Il form per utilizzare i filtri all'interno dello shop

## 4.4 I mixins

All'interno dell'applicazione, nei file `mixins.py`, ho realizzato tutti i controlli per effettuare il controllo d'accesso alle diverse view che compongono il progetto. Ogni mixin che richiede autorizzazioni speciali eredita la classe `LoginRequiredMixin` che controlla che l'utente abbia fatto l'accesso e la classe `UserPassesTestMixin` che richiede di ridefinire la funzione `test_func` in cui fare i diversi controlli necessari per poi ritornare `True` o `False` per dare o meno l'accesso alla view all'utente.

# Capitolo 5

## Test Realizzati

Per i test mi sono concentrato sul verificare che i mixins realizzati si comportassero come atteso e ho voluto testare in modo più esaustivo possibile l'upload di nuovi corsi essendo la funzionalità singola che ha richiesto più lavoro.

### 5.1 Testing delle View

Per ogni test delle view ho utilizzato un utente test che esegue chiamate GET o POST agli url in diverse condizioni, sia a livello di autenticazione che a livello di acquisto di un corso o di possesso dello stesso. Per fare un esempio: nel test `test_teacher_purchase_own_course` l'utente richiede l'acquisto di un corso che lui stesso ha pubblicato, per far sì che il test sia superato eseguo per prima cosa un controllo sullo status code restituito dalla richiesta, che deve essere 200, dopodiché controllo che la pagina ritornata contenga il messaggio d'errore atteso, ovvero quello della pagina che l'apposito mixin si occupa di renderizzare in caso di mancato permesso d'accesso. Alla fine mi sono ritenuto soddisfatto con la realizzazione di 15 test che toccano la maggior parte delle funzioni principali.

```
def test_teacher_purchase_own_course(self):
    """Teachers can't purchase their own course"""
    self.client.login(username='testteacher',
                      password='12345')
    response = self.client.get(
        reverse('shop:course-purchase',
               kwargs={'pk': self.test_course.pk}))

    self.assertEqual(response.status_code, 200)
    self.assertContains(response,
                        "You can't purchase your own course")
```

In generale ogni test segue questo stesso *modus operandi*: vengono creati utenti e altri oggetti di test necessari, si effettua la richiesta da testare e infine si verifica che il mixin abbia fatto i controlli attesi e restituito la pagina giusta

### **5.1.1 Test di creazione di un nuovo corso**

Per la creazione di un nuovo corso ho voluto testare tutti i possibili casi andando a realizzare per ogni test un form "finto" contenente dati sbagliati (e.g.: prezzo negativo, file txt invece di mp4...) da utilizzare poi nella post request per poi controllare infine che il test non venisse creato ma che venisse restituita la pagina contenente la segnalazione dell'errore atteso.

# Capitolo 6

## Risultati ottenuti

### What are functions

Teacher: Itorvalds

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.

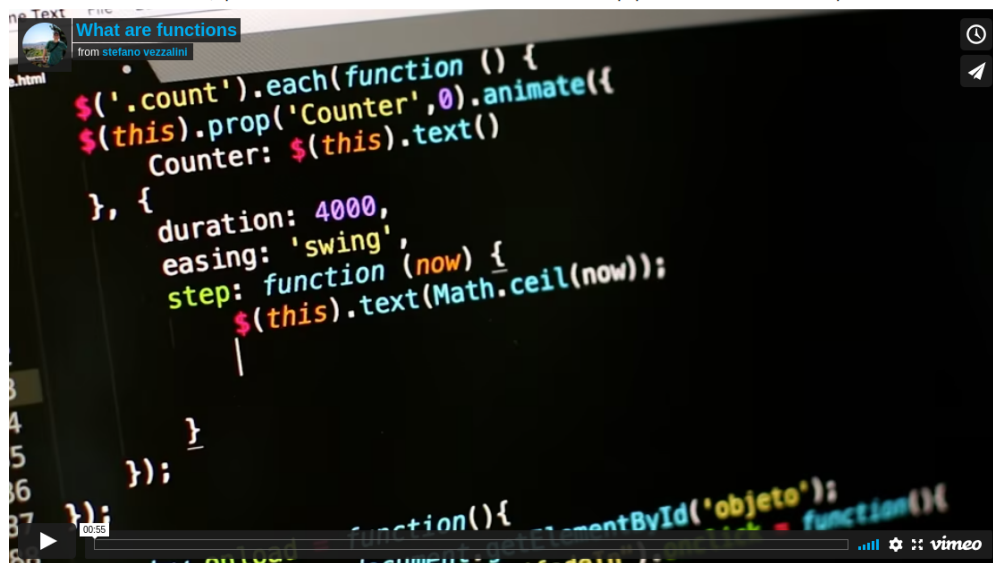


Figura 6.1: Visualizzazione corsi acquistati

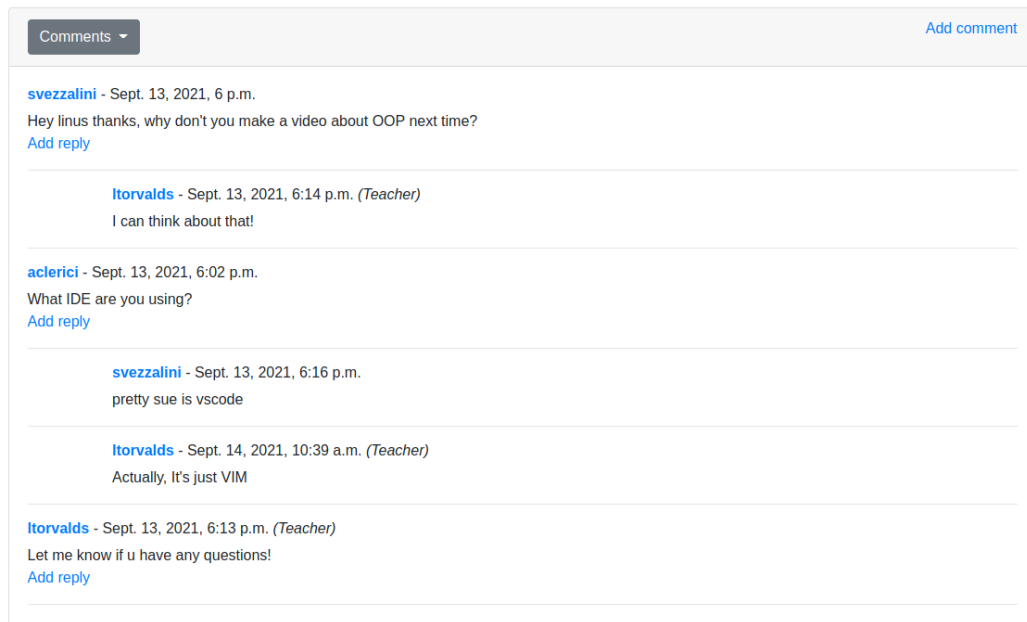


Figura 6.2: Sezione commenti

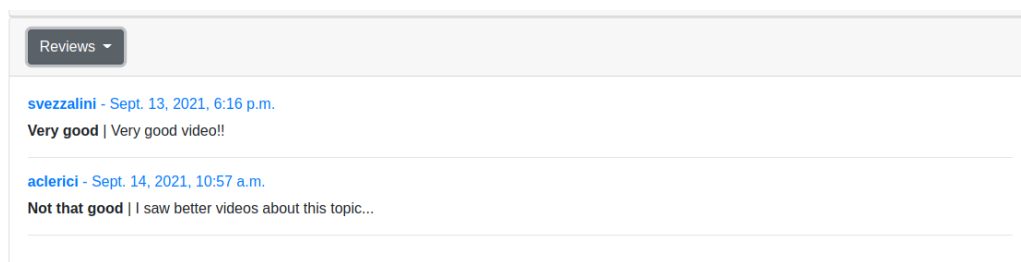



Figura 6.3: Sezione recensioni




**LINUS TORVALDS**  
Email: linuxtorvalds@notgmail.com  
*I will teach you how to code, maybe*  
[Edit Profile](#)

### Published courses

Title	Price	Manage	Total purchases
<a href="#">How to cut peppers</a>	€20.0	<a href="#">Update</a> <a href="#">Delete</a>	0
<a href="#">What are functions</a>	€50.0	<a href="#">Update</a> <a href="#">Delete</a>	2

[New Course](#)

### My courses



Come calciare un pallone

Made by aclerici

[Watch](#)[Add review](#)

Figura 6.4: Profilo insegnante