Student-SharePlace/MoRe Exchange

Progetto di Tecnologie Web – UNIMORE

di Micco Luigi

Descrizione

Il progetto sviluppato consiste un una piattaforma per studenti universitari, nello specifico per tale esame si è voluto prendere come riferimento UNIMORE, da ciò MoRe Exchange.

Questo progetto mira a rendere possibile lo scambio e vendita di appunti, lo scambio di esperienze ed informazioni relative alla propria vita universitaria tra gli studenti dell'Università di Modena e Reggio Emilia.
In particolare uno studente iscrivendosi alla piattaforma, specificando il suo/i suoi Corsi di Laurea (Degree), ha la possibilità di caricare e pubblicare Appunti (Notes).

Gli appunti caricati da uno studente, dovranno essere relativi ad un Corso al quale lo studente è implicitamente iscritto, avendo specificato i suoi CdL.
In tal modo gli appunti saranno visualizzabili dagli altri studenti registrati al portale, ma gli studenti potranno visualizzare solo appunti relativi ai propri Corsi: i corsi che afferiscono ai CdL specificati dagli studenti in fase di registrazione al portale.

Gli appunti caricati da uno studente non sono pienamente accessibili a tutti gli altri studenti iscritti alla piattaforma. Nello specifico gli appunti veri e propri, dovranno consistere in un file, il quale dovrà essere allegato in fase di creazione della risorsa Appunti(Notes) sul portale.

Affinchè uno studente non proprietario possa accedere a tale allegato di appunti non suoi deve:

- aver acquistato tali Appunti dallo studente proprietario (owner)
- oppure, aver effettuato un Scambio con lo studente proprietario di tali appunti, proponendo dei suoi appunti.

Uno scambio affinché possa essere definito tale, deve avere appoggio da entrambi i soggetti i coinvolti:
lo studente che vuole ottenere degli appunti di un altro studente proponendo i suoi deve effettuare una Richiesta di Scambio(Exchange Request), la quale potrà essere accettata o declinata dallo studente ricevente.

Vi è inoltre la possibilità di aggiungere Esperienze(Experiences):
in tal modo uno studente potrà condividere con gli altri studenti iscritti ai suoi Corsi informazioni/esperienze relative alla sfera universitaria.
Basti pensare ad un'esperienza Erasmus, oppure ad un'esperienza avuta ad un esame (es. "Mancanza di un IDE/libreria nei pc del Laboratorio").

Sintentizzando:

- un utente anonimo/non registrato potrà visualizzare la landing page ed iscriversi alla piattaforma

- un utente registrato, quindi uno studente:
  - potrà effettuare operazioni CRUD sui propri Appunti ed Esperienze
  - non potrà modificare o eliminare Risorse (Resources) non proprie
  - modificare il proprio profilo
  - accettare/rifiutare Richieste di Scambio sottopostegli
  - visualizzare gli esiti di richieste di scambio effettuate
  - comprare e successivamente accedere ad appunti di altri studenti

- utente Admin(Django default): tale utente potrà manipolare qualsiasi oggetto e Modello, senza alcun tipo di limitazione, sfruttando il pannello di Admin integrato in Django

Descrizione del progetto dell'applicazione

Diagramma dei Casi d'Uso

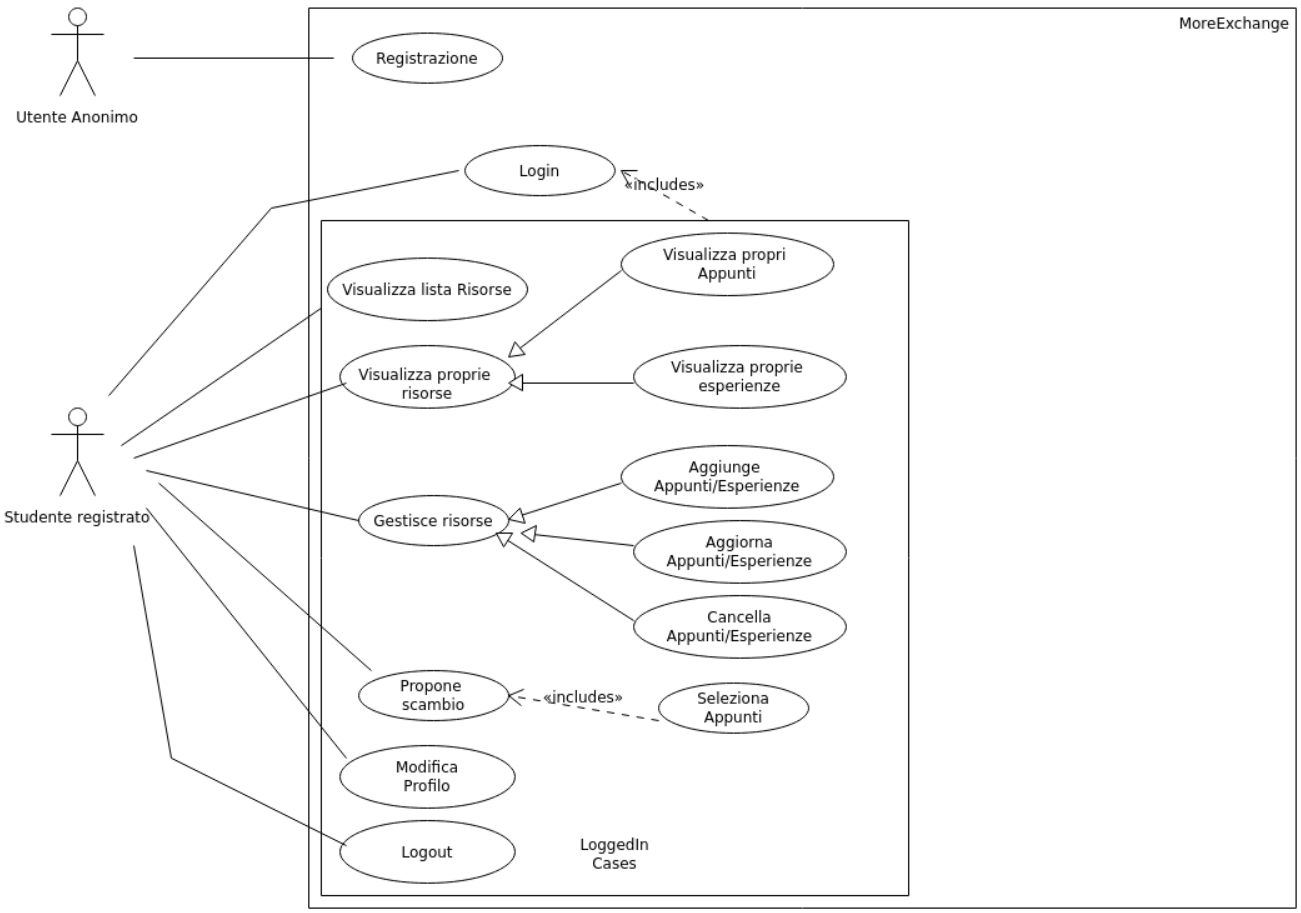
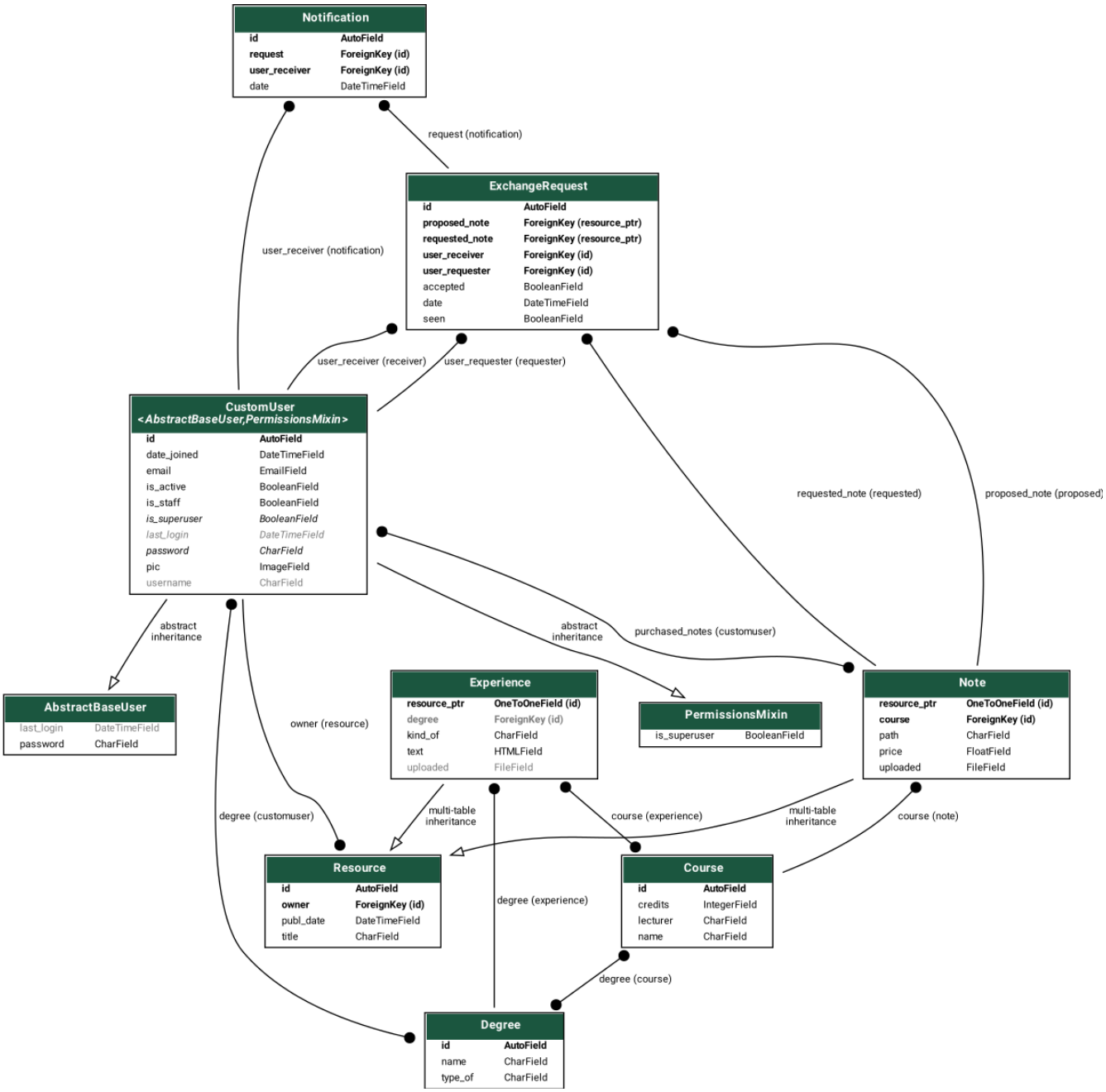
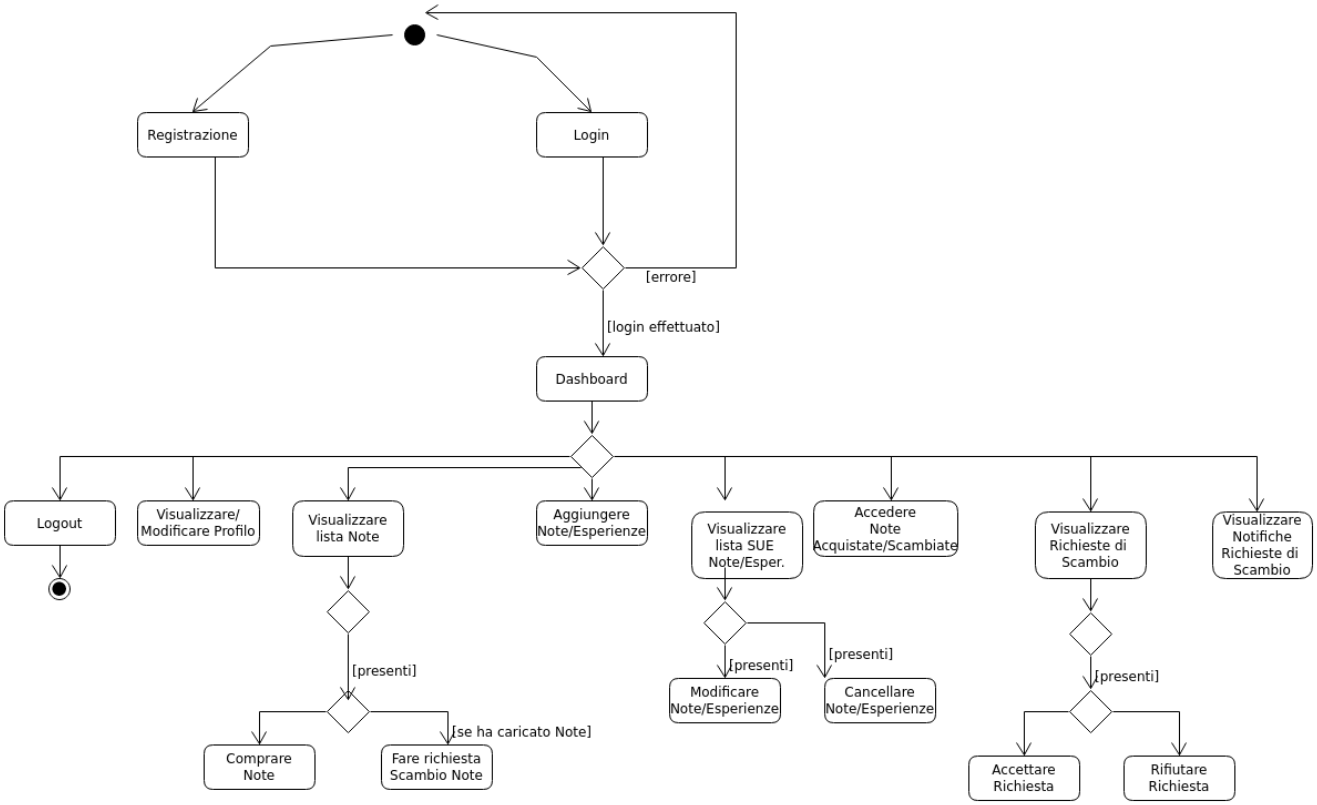


Diagramma delle classi



Tale diagramma rappresenta l’organizzazione e le relazioni tra classi del progetto in esame, fedelissimo alla realtà essendo stato auto-generato utilizzando un’app apposita (django_extension).

Diagramma delle attività



Il diagramma rappresenta tutte le attività che un utente può fare a partire dalla registrazione e quindi login, dopo il quale avrà a disposizione una barra di navigazione che sarà il punto di partenza per tutte le attività effettuabili sul sito.
Il diagramma è piuttosto riassuntivo, al fine di facilitarne la lettura e comprensione.

Tecnologie Utilizzate

L'applicazione Web è stata realizzata attraverso l'utilizzo del web framework Django, con app aggiuntive per la gestione di funzionalità particolari in piena filosofia DRY (di Django):

- crispy-forms: per rendere responsive e più user-friendly i vari Form presenti
- tiny-mce: per rendere disponibile un editor di testo Rich Text online.

Il backend usato per il database è SQLite e i dati sono salvati in locale, essendo il default (in caso di un utilizzo meno didattico, è necessaria una migrazione immediata a PostGreSQL).
Lato client è stato utilizzato un template (Argon Dashboard) basato su Bootstrap e quindi Html, Css (con vari preprocessori: Sass in primis), JS e la libreria Jquery.

Organizzazione logica dell’applicazione

L’applicazione è strutturata in due app:

- users
- contents.

Nell’app users viene gestita la parte relativa agli account utente, è stato definito un CustomUser come modello per l’utente per ottenere una maggiore flessibilità nella gestione dei campi obbligatori e non.
Sono quindi presenti in tale app, viste e template per la registrazione, login e modifica del profilo utente.
Nell’app contents sono presenti le varie classi per gestione dei Contenuti che un utente può caricare, come già menzionati più volte: Appunti ed Esperienze.
Ma sono presenti tutti i modelli e le funzionalità per la gestione di contorno di queste due Risorse principali, quindi oltre alle viste per Accedere, Modificare, Creare ed Eliminare le risorse; sono qui collocati i componenti per la gestione delle richieste di Scambio e compera delle Note.

Una scelta singolare e piuttosto discutibile è l’aver collocato nell’app utente, viste e template non strettamente relativi alla gestione degli utenti, ma piuttosto in mezzo agli utenti ed i contenuti:
ho scelto di lasciare in tale app le view ed i template che permettono di consultare le Esperienze e gli Appunti caricati da un certo utente, ossia che permettono ad un utente di visualizzare le proprie risorse.

Nell’implementazione ho prediletto l’utilizzo di Class Based Views, sia per sfruttare appieno tale approccio DRY di Django che per imparare ad affrontare un progetto di modeste dimensioni, orientato quasi completamente ad oggetti, sfruttando quindi in alcuni casi Mixin già presenti in Django oppure da me impostati.

In altri casi, in particolare nell view richiamate attraverso AJAX ho utilizzato Function Based Views più congeniali all’utilizzo.

Test effettuati

I test funzionali implementati ed effettuati sono collocati nel file tests.py rispettivamente all'interno dei packages users e contents.
Mi sono concentrato principalmente sul testing di View.
Per quanto riguarda l'app users, oltre ad aver testato i CustomManagers da me implementati per l'utilizzo di un CustomUser a partire di un AbstractBaseUser, ho testato in modo abbastanza completo le View di Registrazione e Login, in un caso ho voluto testare la View Login subito successivamente alla creazione di uno user usando sempre un test.

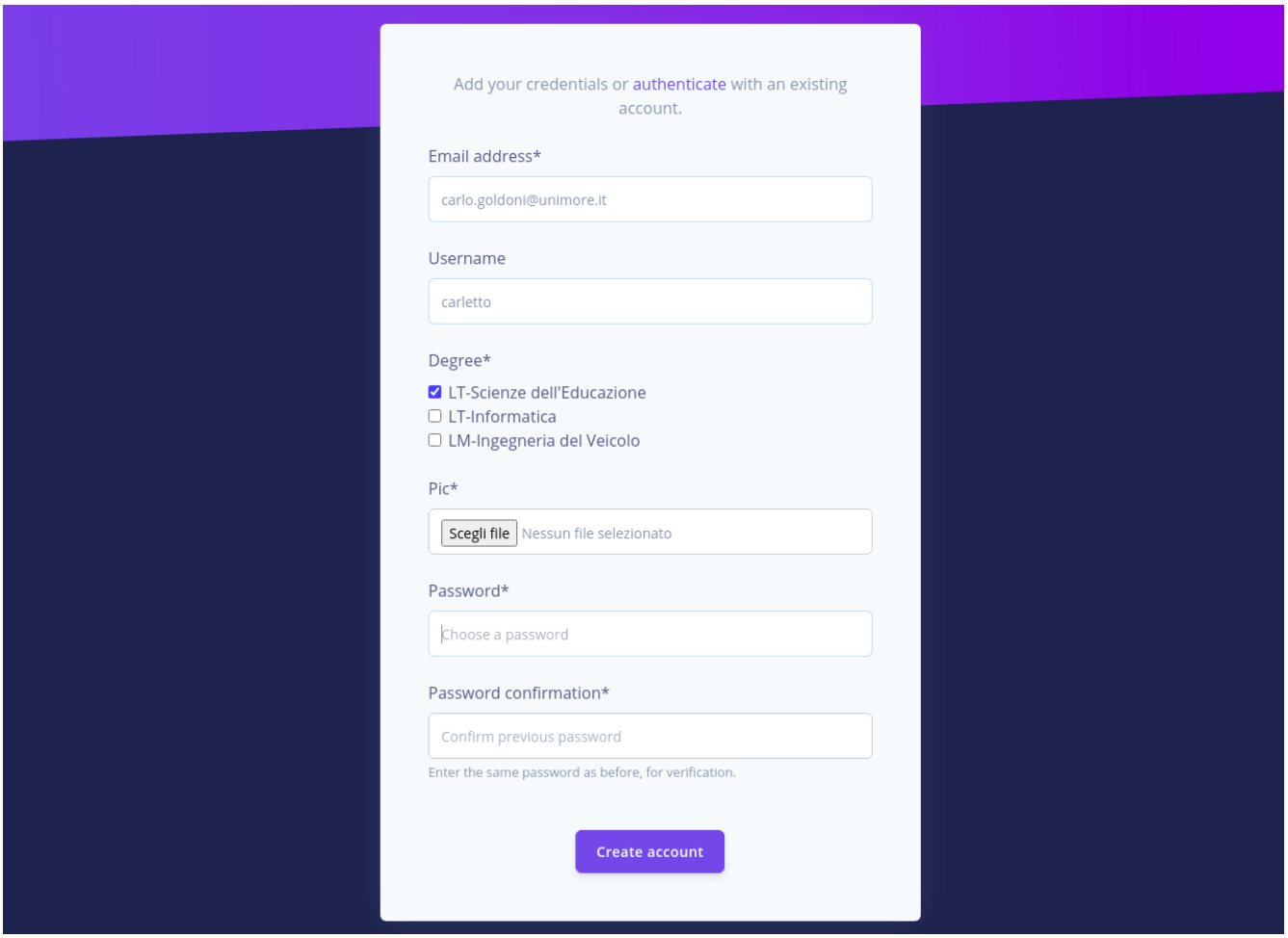
```python
def Test_user_login_view_POST_success(self):
    """
    Test che si occupa di verificare che dato l'url del login utente, effettuata una richiesta POST con
    i dati dell'utente si venga rediretti alla homepage (login effettuato correttamente)

    :return: None
    """
    response = self.client.post(self.login_url, {
        'username': 'myemail@email.it',
        'password': 'pwtest12'
    }, follow=True)

    # get user registered with previous successful test
    user_in_db = CustomUser.objects.get(email='myemail@email.it')

    self.assertRedirects(response, reverse('content:course-list'), status_code=302,
                         target_status_code=200)  # redirection to homepage
    self.assertTemplateUsed(response, 'course_tables.html')  # so will be displayed another time login page
    self.assertEquals(response.context['user'], user_in_db)  # so correct user is logged

def test_user_login_view_POST_error(self):
    """
    Test che si occupa di verificare che dato l'url del login utente, effettuata una richiesta POST con
    i dati dell'utente (errati) non si venga rediretti alla homepage ma si visualizzi la schermata di login
    con gli errori specificati (login fallito)

    :return: None
    """

    response = self.client.post(self.login_url, {
        'email': 'notregistered@email.it',
        'password': 'unusefulpwd'
    })
    self.assertEqual(response.status_code, 200)  # no redirect to homepage
    self.assertTemplateUsed(response, 'registration/login.html')  # so will be displayed another time login page
    self.assertEquals(response.context['user'].pk, AnonymousUser.pk)  # and user is Anonymous

def test_user_registration_and_login_no_session(self):
    """
    Test generale che concatena una registrazione avvenuta correttamente, non mantenendo l'utente loggato,
    con un login successivo dell'utente appena registrato
    :return: None
    """
    self.Test_user_registration_view_POST_success_redirect()
    self.client.logout()  # delete session created with registration
    self.Test_user_login_view_POST_success()
```

Per quanto riguarda i test nell'app contents ho scelto di testare View, una Function-Based-View, che coinvolgesse la stragrande maggioranza dei modelli utilizzati in tale app:
la FBV manage_exchange, la quale viene chiamata attraverso AJAX.
Questa si occupa di gestire l'accettazione o rifiuto di una richiesta di scambio, così sono stati testati entrambi i casi (Accept/Reject), dopo aver effettuato il SetUp ossia aver predisposto gli oggetti necessari a soddisfare i vincoli relazionali della ExchangeRequest.

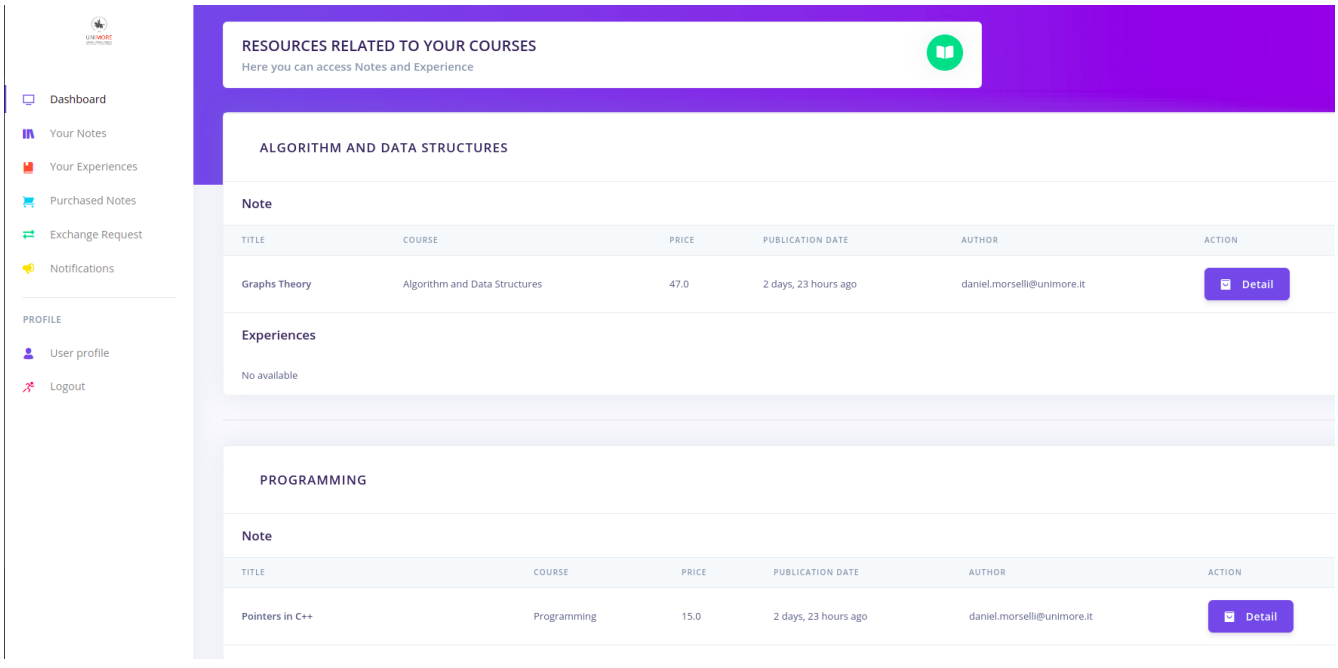
```
class TestViews(TestCase):

    def setUp(self):
        self.client = Client()
        self.url_action = reverse('content:request-action')
        self.login_url = reverse('login')
        self.d = Degree.objects.create(name="Biotech", type_of='LT')
        self.c = Course.objects.create(name="Bioinformatics", credits=3, lecturer='Bicciato')
        self.c.degree.add(self.d)
        self.user_1 = CustomUser.objects.create_user(email="leopoldo@unimore.it", username="leopoldo", password="pass")
        self.user_1.degree.add(self.d)
        self.user_2 = CustomUser.objects.create_user(email="gianbattista@unimore.it", username="gianbattista",
                                                     password="pass")
        self.user_2.degree.add(self.d)
        self.n1 = Note.objects.create(title='Note 1', owner=self.user_1, course=self.c, price=12)
        self.n2 = Note.objects.create(title='Note 2', owner=self.user_2, course=self.c, price=10)
        self.er = ExchangeRequest.objects.create(requested_note=self.n2, proposed_note=self.n1,
                                                 user_requester=self.n1.owner,
                                                 user_receiver=self.n2.owner)

    def test_manage_exchange_view_REJECT(self):
        self.assertFalse(self.er.seen)
        self.assertFalse(self.er.accepted)
        self.assertTrue(self.client.login(username="leopoldo@unimore.it", password="pass"))
        response = self.client.post(self.url_action, {
            'req': self.er.id,
            'action': 'reject',
        }, follow=True)
        refreshed_er = ExchangeRequest.objects.get(id=self.er.id)
        self.assertEqual(response.status_code, 200)  # no redirect to homepage
        self.assertFalse(refreshed_er.accepted)
        self.assertTrue(refreshed_er.seen)
        self.assertGreaterEqual(Notification.objects.filter(request__id=self.er.id).count(), 1)

    def test_manage_exchange_view_ACCEPT(self):
        self.assertFalse(self.er.seen)
        self.assertFalse(self.er.accepted)
        self.assertTrue(self.client.login(username="leopoldo@unimore.it", password="pass"))
        response = self.client.post(self.url_action, {
            'req': self.er.id,
            'action': 'accept',
        }, follow=True)
        refreshed_er = ExchangeRequest.objects.get(id=self.er.id)
        self.assertEqual(response.status_code, 200)  # no redirect to homepage
        self.assertTrue(refreshed_er.accepted)
        self.assertTrue(refreshed_er.seen)
        self.assertGreaterEqual(Notification.objects.filter(request__id=self.er.id).count(), 1)
```

Risultati progettuali

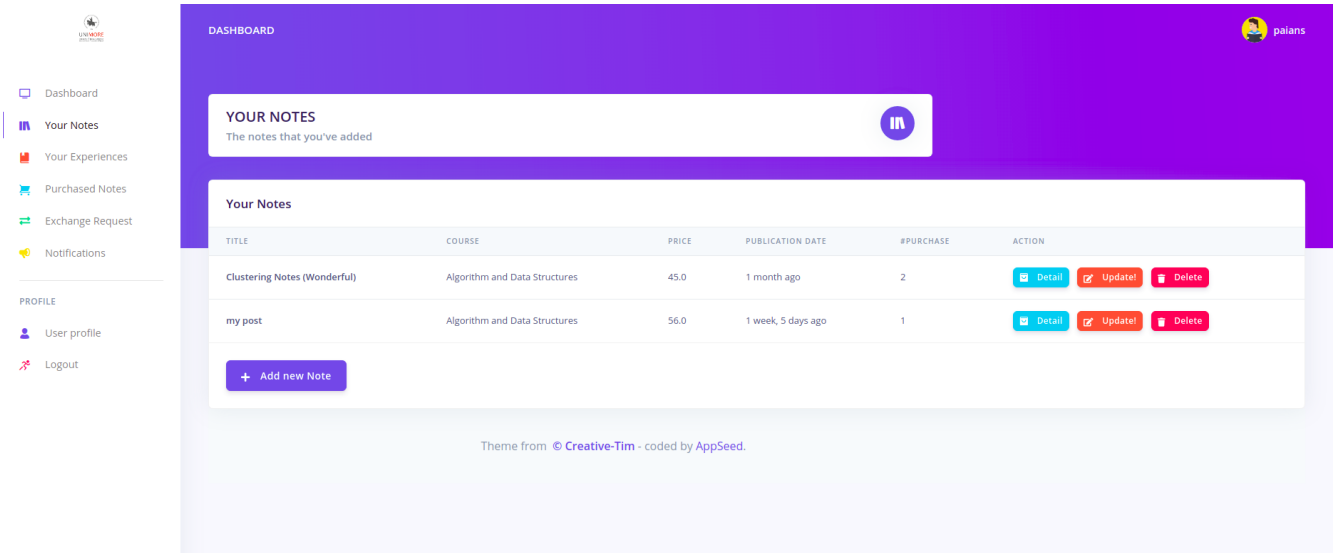
Pagina di registrazione



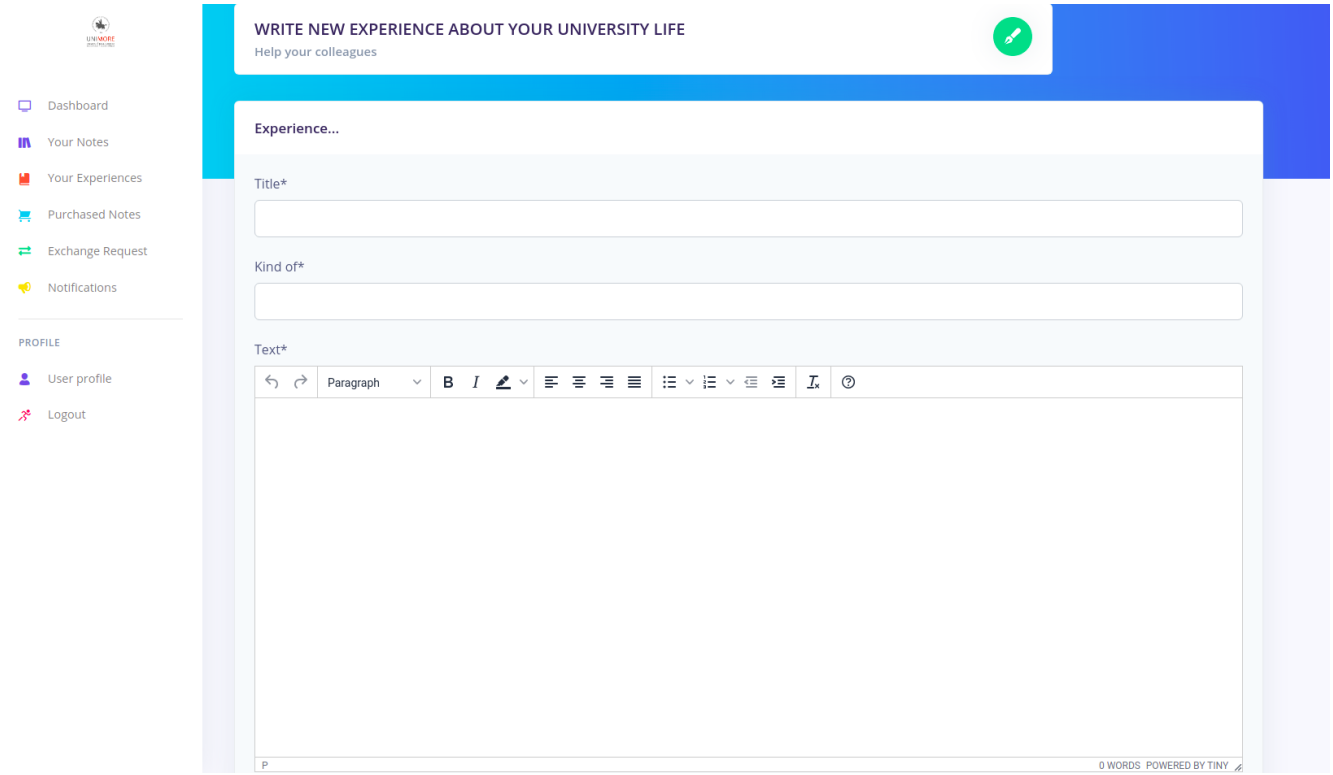
Dashboard: Appunti ed esperienze visualizzati per Corsi (dell’utente loggato)



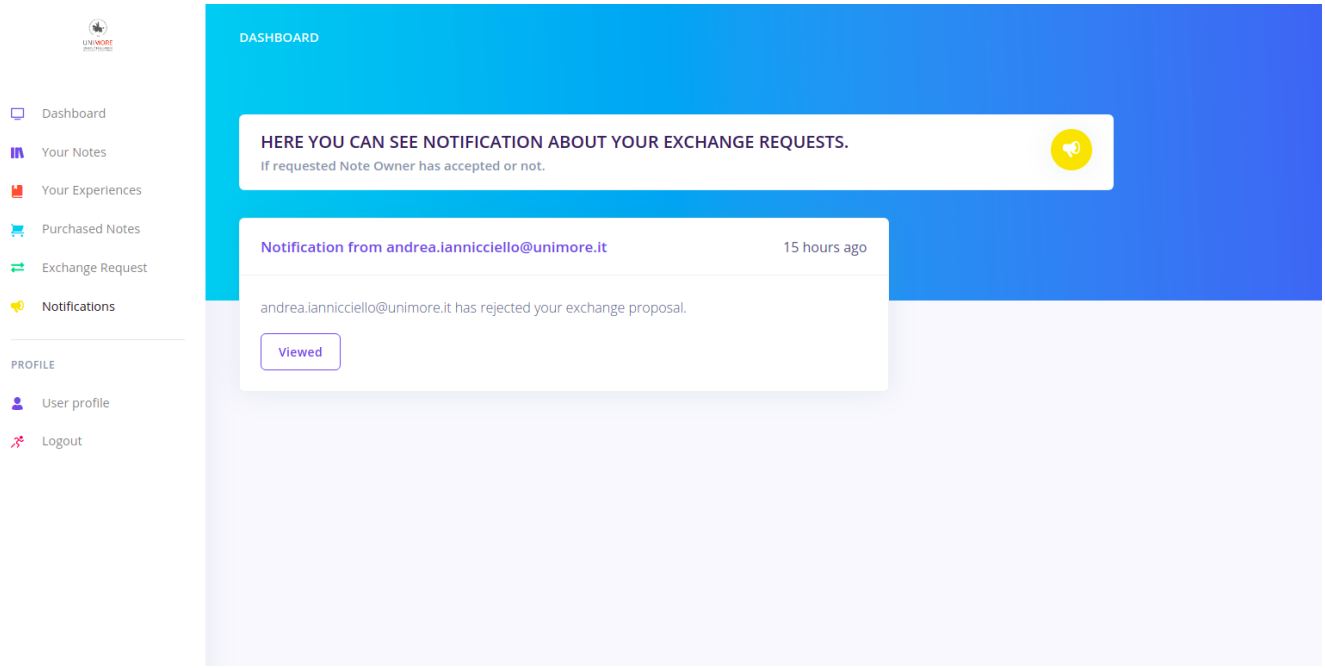
Elenco note dell’utente



Pagina di creazione di una nuova esperienza



Pagina di visualizzazione notifiche



Problemi riscontrati

Durante lo sviluppo del progetto ho incontrato diversi problemi che mi hanno fatto perdere diverse giornate di lavoro:

- avrei voluto aggiungere la possibilità di commentare le esperienze: ho speso diversi giorni a studiare e cercare di utilizzare l'app open-source django-comments-xtd. Tuttavia vi sono stati una serie di problemi che non facevano funzionare tale app, all'interno del mio progetto, sicuramente la documentazione abbastanza scarsa non ha aiutato;
- la gestione dei permessi degli utenti: leggendo sui vari StackOverflow, ho potuto constatare l'assenza di un sistema di gestione dei permessi ROW-LEVEL/Object-Level in Django. Mi sono quindi imbattuto in diverse librerie (Rules, Django-Oso/Oso), le quali anche mi hanno portato via molto tempo e alla fine vi erano problemi nell'utilizzo, sebbene le premesse sembrassero buone(documentazione abbastanza diretta e comprensibile, Star on GitHub)
- l'utilizzo di un widget per mostrare un anteprima dell'immagine scelta come immagine di profilo, ho lavorato ad un componente lato JS, tuttavia nel modo in cuoi l'avevo interfacciato con Django pare ci fosse qualche problema visto che il caricamento dell'immagine non andava a buon fine.