
django
REST
framework 3

UNIMORE
UNIVERSITÀ DEGLI STUDI DI MODENA E REGGIO EMILIA

API

Un API (Application Programming Interface) è uno strumento formale per descrivere la comunicazione diretta tra diversi sottosistemi.

Esistono diversi tipi di API (anche le interfacce rientrano nella definizione) ma, nel caso di comunicazione tra sistemi remoti, le più utilizzate sono le Web API.

Le Web API permettono di trasferire i dati via Web (facendo la richiesta di un certo servizio ad un URL) e sono strutturate secondo un pattern RESTful (REpresentational State Transfer).

django REST framework

UNIMORE
UNIVERSITÀ DEGLI STUDI DI MODENA E REGGIO EMILIA

perché le API?

Il vantaggio di un approccio *API-first* è la separazione netta tra backend (che si occupa di gestire ed elaborare i dati) e frontend (che si occupa di visualizzarli).

In questo modo si rendono accessibili gli stessi dati e gli stessi servizi ad applicazioni diverse (Web, mobile, desktop) usando linguaggi o framework diversi, senza dover riscrivere ogni volta la parte di backend.

Un API può essere utilizzata sia internamente che esternamente al sistema sviluppato.

Per esempio Google mette a disposizione per gli sviluppatori diverse API (autenticazione, utilizzo delle mappe di Maps, Drive, etc).

django REST framework

UNIMORE
UNIVERSITÀ DEGLI STUDI DI MODENA E REGGIO EMILIA

urls

URL (Uniform Resource Locator) è l'indirizzo per trovare una risorsa nel Web.

Attraverso un URL il client fa una richiesta (**request**) al server che a sua volta invia una risposta (**response**).

L'URL è formato da:

- protocollo (http, https, ftp, smtp, ecc.)
- hostname (www.website.com)
- percorso opzionale (/about/, /contacts/, ecc.)

UNIMORE
UNIVERSITÀ DEGLI STUDI DI MODENA E REGGIO EMILIA

metodi HTTP

Poiché il Web è diventata la *killer application* di Internet, HTTP (HTTPS nella versione sicura) è il protocollo di comunicazione più utilizzato nella comunicazione tra client e server.

HTTP definisce una serie di metodi che possono essere utilizzati nell'interazione con un server. I più noti sono `POST`, `GET`, `PUT` e `DELETE` e corrispondono alle azioni `CRUD` (`create`, `read`, `update` e `delete`).

Per creare un elemento si usa `POST`, per leggerlo `GET`, per modificarlo `PUT` e per cancellarlo `DELETE`.

UNIMORE
UNIVERSITÀ DEGLI STUDI DI MODENA E REGGIO EMILIA

endpoints

Un sito Web è composto da pagine Web scritte in HTML che possono contenere CSS, Javascript, immagini, video e altro.

Una Web API ha degli *endpoints* che sono degli URLs associati ad una serie di possibili azioni (metodi http) che espongono dei dati (tipicamente in formato JSON, o in alternativa in XML).

Esempi:

- https://www.website.com/api/users/
  # `GET` restituisce la lista degli utenti, `POST` permette di creare un nuovo utente
- https://www.website.com/api/users/1/
  # `GET` restituisce un singolo utente, `PUT` permette di modificarlo e `DELETE` di eliminarlo

django REST framework

UNIMORE
UNIVERSITÀ DEGLI STUDI DI MODENA E REGGIO EMILIA

messaggi HTTP

La comunicazione tra client e server è descritta dai *messaggi HTTP*.

Ogni messaggio HTTP contiene *status line* (request/response), *headers* e *body* (opzionale).

Esempio messaggio HTTP di richiesta:

```
GET / HTTP/1.1
Host: google.com
Accept_Language: en-US
```

La prima riga è nota come *request line* e specifica il metodo HTTP da utilizzare (`GET`), il percorso (`/`) e la versione http (`HTTP/1.1`).

messaggi HTTP

Esempio messaggio HTTP di risposta:

```
HTTP/1.1 200 OK
Date: Wed, 17 Feb 2020 23:26:07 GMT
Server: gws
Accept-Ranges: bytes
Content-Length: 13
Content-Type: text/html; charset=UTF-8

Hello world!
```

La prima riga è nota come *response line* e specifica la versione http (`HTTP/1.1`) e lo stato della risposta (`200 OK`). Dopo il *line break* c'è il contenuto della risposta (`Hello world!`).

UNIMORE
UNIVERSITÀ DEGLI STUDI DI MODENA E REGGIO EMILIA

messaggi HTTP

In generale:

```
response/request line
headers…


(optional) body
```

La maggior parte delle pagine Web contiene diverse risorse (HTML, CSS, immagini, ecc.), per ognuna delle quali occorre un ciclo di richiesta/risposta HTTP prima che il browser renderizzi completamente la pagina.

status code

Per capire se la richiesta del client o la risposta del server hanno avuto successo, HTTP usa degli *status codes* :

- 2** `Success`, l'azione richiesta dal client è stata ricevuta, capita e accettata.
- 3** `Redirection`, l'URL richiesto è stato spostato.
- 4** `Client Error`, c'è stato un errore nella richiesta del client (tipicamente ha sbagliato l'URL).
- 5** `Server Error`, il server ha avuto problemi nella risoluzione della richiesta.

I più noti sono: `200 (OK)`, `201 (Created)`, `301(Moved Permanently)`, `404 (Not Found)` e `500 (Server Error)`.
Lo *status code* si trova nella *request/response line* del messaggio HTTP.

statelessness

Il protocollo HTTP non prevede uno stato, ovvero ogni coppia request/response è indipendente dalla precedente.

Questo permette di far fronte a perdite di segnale tra un ciclo di request/response e l'altro, rendendo il protocollo HTTP molto resistente.

Avere uno stato tuttavia è molto importante. Permette per esempio di salvare informazioni come l'autenticazione dell'utente o il suo carrello su un sito di ecommerce.

Lo stato viene tipicamente salvato lato client, nel browser dell'utente (sessioni e cookie).

UNIMORE
UNIVERSITÀ DEGLI STUDI DI MODENA E REGGIO EMILIA

REST

REpresentational State Transfer (REST) è un'architettura che permette di costruire web API.

Ogni RESTful API ha 3 caratteristiche principali:

- è stateless, come HTTP
- supporta i più noti metodi HTTP (`GET`, `POST`, `PUT`, `DELETE`)
- restituisce dati in formato JSON o XML

django REST framework

UNIMORE
UNIVERSITÀ DEGLI STUDI DI MODENA E REGGIO EMILIA

Django REST Framework

Django REST Framework è un'applicazione Django che permette di integrare al framework tradizionale la gestione delle web API.

Può essere utilizzato solo all'interno di un progetto Django e si installa col comando:

```
pip install djangorestframework
```

Essendo un'applicazione Django a tutti gli effetti, deve essere aggiunta alla lista `INSTALLED_APPS` del file `settings.py` del progetto Django:

```
INSTALLED_APPS = [
    ...
    'rest_framework',
]
```

UNIMORE
UNIVERSITÀ DEGLI STUDI DI MODENA E REGGIO EMILIA

Django REST Framework

Creiamo una nuova applicazione per la gestione delle web api:
`python manage.py startapp api`
e aggiungerla alla lista `INSTALLED_APPS` di `settings.py`.

```
INSTALLED_APPS = [
    ...
    ‘rest_framework’,
    ‘api.apps.ApiConfig’,
]
```

Questa applicazione non deve contenere i modelli per mappare il database.
Si occupa di serializzare i dati, renderli accessibili attraverso gli URL e mostrarli attraverso le views.

django REST framework

UNIMORE
UNIVERSITÀ DEGLI STUDI DI MODENA E REGGIO EMILIA

serializers

Il *serializer* permette di convertire un oggetto complesso (tipicamente un oggetto della classe Model) in uno più semplice, convertendolo in formato json, in modo che possa essere leggibile dagli endpoints delle api.

Si crea un file `serializers.api` all'interno dell'applicazione `api`.

```
from rest_framework import serializers
from books.models import Book

class BookSerializer(serializers.Serializer):
    class Meta:
        model = Book
         fields = ('id', 'title', 'description')
```

django REST framework

UNIMORE
UNIVERSITÀ DEGLI STUDI DI MODENA E REGGIO EMILIA

views

All'interno del file `views.py` dell'applicazione `api`:

```
from rest_framework import generics

from books.models Book
from .serializers import BookSerializer

class BookListAPIView(generics.ListAPIView):
    queryset = Book.objects.all()
    serializer_class = BookSerializer
```

Creiamo una view che renda accessibili, in formato json, gli oggetti specificati nell'attributo `queryset` di `BookAPIView`, serializzandoli con la classe `BookSerializer`.

django REST framework

UNIMORE
UNIVERSITÀ DEGLI STUDI DI MODENA E REGGIO EMILIA

urls

All'interno del file `urls.py` del progetto:

```
from django.contrib import admin
from django.urls import path, include

urlpatterns = [
    path('admin/', admin.site.urls),
    path('api/v1/', include('api.urls'),
]
```

Includiamo gli urls dell'applicazione `api` e li associamo all'indirizzo `/api/v1/` ("v1" sta per "versione 1". Le api possono subire delle modifiche ma è bene che le versioni precedenti continuino a funzionare).

UNIMORE
UNIVERSITÀ DEGLI STUDI DI MODENA E REGGIO EMILIA

urls

All’interno del file `urls.py` dell’applicazione `api`:

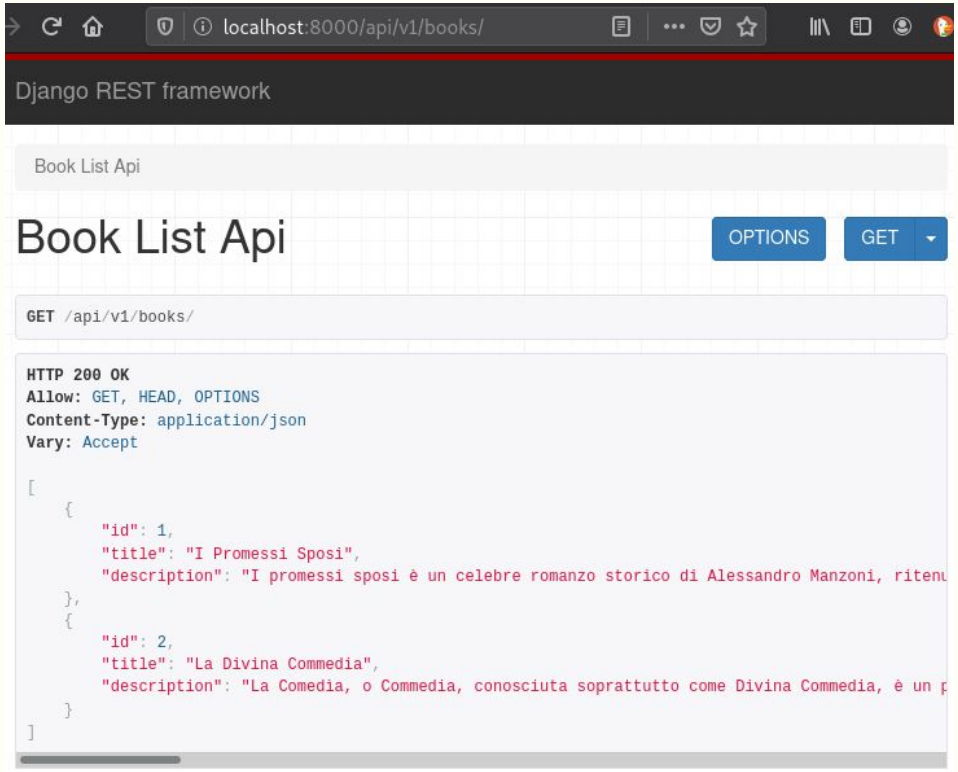
```
from django.urls import path
from .views import BookListAPIView

urlpatterns = [
    path(‘books/’, BookListAPIView.as_view()),
]
```

Creiamo l’endpoint per accedere all’api che mostra la lista dei libri in formato json.

interfaccia

`python manage.py runserver` e ...



Django REST Framework di default implementa questa interfaccia grafica.

Cliccando su GET possiamo scegliere se visualizzare i dati in questo modo o solamente in json.

UNIMORE
UNIVERSITÀ DEGLI STUDI DI MODENA E REGGIO EMILIA

accesso alle API

L’obiettivo delle API è quello di rendere indipendenti la parte di backend, che elabora e gestisce i dati, e quella di frontend, che li mostra e li rende interagibili.

Per rendere accessibili le API ad altre applicazioni, tipicamente sviluppate con framework JavaScript come React, Angular o Vue, è necessario dare accesso ai loro domini.

Per farlo si utilizza il CORS (Cross-origin resource sharing), un meccanismo che restringe l’accesso ad una risorsa su una pagina web attraverso una whitelist.

UNIMORE
UNIVERSITÀ DEGLI STUDI DI MODENA E REGGIO EMILIA

CORS

Si installa l'applicazione django che permette di gestire il meccanismo CORS:

```
pip install django-cors-headers
```

Si aggiunge l'applicazione alla lista `INSTALLED_APPS` del file `settings.py` del progetto django:

```
INSTALLED_APPS = [
    ...
    'rest_framework',
    'corsheaders',
    'api.apps.ApiConfig',
]
```

django REST framework

UNIMORE
UNIVERSITÀ DEGLI STUDI DI MODENA E REGGIO EMILIA

CORS

Si aggiunge il middleware alla lista `MIDDLEWARE` in `settings.py`:

```
MIDDLEWARE = [
    ...
    'corsheaders.middleware.CorsMiddleware',
    'django.middleware.common.CommonMiddleware',
    ...
]
```

È importante che sia inserito nella lista prima di `CommonMiddleware` perché i `MIDDLEWARE` sono caricati dall'alto verso il basso e devo rispettare un certo ordine.

UNIMORE
UNIVERSITÀ DEGLI STUDI DI MODENA E REGGIO EMILIA

CORS

Si aggiunge la lista dei domini che hanno accesso alle risorse API in `settings.py`:

```
CORS_ORIGIN_WHITELIST = [
    ‘http://localhost:8000’,
    ‘http://localhost:3000’,
]
```

Il primo indirizzo ha come porta 8000, che è quella usata di default da Django.
Il secondo indirizzo ha come porta 3000, che è quella usata di default da React.

UNIMORE
UNIVERSITÀ DEGLI STUDI DI MODENA E REGGIO EMILIA

tipi di views

Come abbiamo visto, i metodi HTTP più noti corrispondono alle funzionalità CRUD (`CREATE`, `READ`, `UPDATE` e `DELETE`).

Questi metodi vengono utilizzati dall'applicazione che si vuole sviluppare, attraverso le APIView di Django REST Framework:

- `ListAPIView`: restituisce un json con la lista degli oggetti serializzati.
- `CreateAPIView`: permette di creare un nuovo oggetto.
- `RetrieveAPIView`: restituisce un json del dettaglio di un singolo oggetto serializzato.
- `UpdateAPIView`: permette di modificare un oggetto già esistente.
- `DestroyAPIView`: permette di eliminare un oggetto.

Django REST framework

tipi di views

Si implementano tutte allo stesso modo, ovvero specificando gli attributi di classe `queryset` e `serializer_class`.
Inoltre presentano a gruppi pattern molto simili:

- `ListAPIView` e `CreateAPIView` nell'url associato non hanno bisogno di specificare l'identificativo dell'oggetto.
- `RetrieveAPIView`, `UpdateAPIView` e `DestroyAPIView` nell'url associato hanno bisogno dell'identificativo dell'oggetto.

A livello di view quindi hanno tutti la stessa implementazione, a livello di url invece differiscono per presenza (o meno) dell'identificativo.

UNIMORE
UNIVERSITÀ DEGLI STUDI DI MODENA E REGGIO EMILIA

viewsets

Per evitare di scrivere più volte lo stesso codice, Django REST Framework mette a disposizione i *viewsets*, cioè delle classi che integrano tutte le funzioni CRUD.

```
from rest_framework import viewsets

from books.models Book
from .serializers import BookSerializer

class BookViewSet(viewsets.ModelViewSet):
    queryset = Book.objects.all()
    serializer_class = BookSerializer
```

django REST framework

UNIMORE
UNIVERSITÀ DEGLI STUDI DI MODENA E REGGIO EMILIA

routers

È possibile risparmiare il codice anche a livello di url, usando i *routers*.

```
from django.urls import path
from rest_framework import routers
from .views import BookViewSet

router = SimpleRouter()
router.register('books', BookViewSet, basename='books')
urlpatterns = router.urls
```

`base_name` è il parametro corrispettivo del parametro `name` di `path`.

In questo modo non è necessario specificare i singoli urls corrispondenti alle view.

UNIMORE
UNIVERSITÀ DEGLI STUDI DI MODENA E REGGIO EMILIA

permessi

È possibile impostare dei permessi per accedere alle API, in modo che solo alcuni utenti possano accedervi.

Possono essere impostati a livello di:

- Project, i permessi si applicano a tutte le api del progetto
- App, i permessi si applicano alle api di una certa app (?)
- View, i permessi si applicano solo all'api di una certa view

Oltre ai permessi già integrati, se ne possono creare di nuovi.

permessi (progetto)

Nel file `settings.py` del progetto si aggiunge il dizionario `REST_FRAMEWORK`:

```
REST_FRAMEWORK = {
    ‘DEFAULT_PERMISSION_CLASSES’: [
        ‘rest_framework.permissions.IsAuthenticated’,
    ]
}
```

Il permesso che abbiamo aggiunto è `IsAuthenticated`, ovvero un controllo che rende le api accessibile solo agli utenti loggati.
Per rendere le api utilizzabili anche dagli utenti non loggati si usa `AllowAny`.

UNIMORE
UNIVERSITÀ DEGLI STUDI DI MODENA E REGGIO EMILIA

permessi (view)

Nel file `views.py` dell'applicazione, impostiamo l'attributo di classe `permission_classes` di una `APIView`:

```
from rest_framework import generics
from books.models Book
from .serializers import BookSerializer

class BookListAPIView(generics.ListAPIView):
    permission_classes = (permissions.IsAuthenticated, )
    queryset = Book.objects.all()
    serializer_class = BookSerializer
```

In questo modo l'api associata a `BookListAPIView` è accessibile solo agli utenti loggati.

django REST framework

UNIMORE
UNIVERSITÀ DEGLI STUDI DI MODENA E REGGIO EMILIA

permessi personalizzati

È buona abitudine creare un file `permissions.py` dentro l'applicazione:

```
from rest_framework import permissions

class IsAuthorOrReadOnly(permissions.BasePermission):

    def has_object_permission(self, request, view, obj):
        # SAFE_METHODS include i metodi HTTP di sola lettura
        if request.method in permissions.SAFE_METHODS:
            return True

        # altrimenti controlliamo che l'oggetto sia stato creato
        # dall'utente che fa la richiesta
        return obj.created_by == request.user
```

django REST framework

autenticazione

HTTP è un protocollo stateless, quindi non ha già integrata una funzionalità per ricordare che l'utente sia autenticato.

Esistono 3 tipi di autenticazioni:

- Autenticazione Base
- Autenticazione di Sessione
- Autenticazione con token

autenticazione base

Quando il client fa una richiesta HTTP al server, deve inviare delle credenziali di autenticazione approvate prima che l'accesso sia garantito:

1. Il client fa una richiesta HTTP.
2. Il server risponde con una risposta HTTP con stato `401 (Unauthorized)` e con un header `WWW-Authenticate`.
3. Il client invia le credenziali contenute nell'header `Authorization`.
4. Il server controlla le credenziali e risponde con status `200 OK` o `403 Forbidden`.

Una volta ricevuta l'approvazione, il client invia tutte le future richieste usando le credenziali dell'header `Authorization`.

autenticazione base

Le credenziali per l'autorizzazione vengono mandate in chiaro e codificate in base64, formattate in questo modo: `<username>:<password>`.

Vantaggi:

- Semplice

Svantaggi:

- Inefficiente: ogni richiesta deve essere validata dal server che controlla username e password.
- Non sicuro: le credenziali sono passate in chiaro.

Questo tipo di autenticazione ha senso solo se si utilizza HTTPS.

autenticazione di sessione

È possibile salvare i dati in variabili temporanee: sessioni e cookies.

Il client si autentica con username e password e riceve un *ID di sessione* dal server, che lo salva come cookie. L'ID di sessione viene passato nell'header in ogni futura richiesta HTTP.

Quando il server legge questo ID, lo usa per accedere ad un oggetto di sessione contenente tutte le informazioni relative al corrispettivo utente, tra cui le credenziali.

Questo approccio è *stateful*, perché viene conservata un'informazione sia lato client (ID di sessione), sia lato server (oggetto di sessione).

autenticazione di sessione

1. Il client inserisce le proprie credenziali (tipicamente attraverso un form).
2. Il server verifica che le credenziali siano corrette e genera un oggetto di sessione salvato sul DB.
3. Il server invia al client un ID di sessione, che viene salvato come cookie nel browser.
4. In tutte le future richieste l'ID di sessione viene incluso nell'header HTTP e, se verificato dal DB del server, la richiesta procede.
5. Quando l'utente fa logout, l'ID di sessione viene distrutto sia lato client che lato server.
6. Se l'utente accede nuovamente, allora viene generato un ID di sessione e salvato come cookie dal client.

autenticazione con token

È un tipo di autenticazione *stateless* : il client invia le proprie credenziali al server, poi viene generato un token univoco e salvato nei cookie o nel local storage.

Il token viene passato nell'header di ogni richiesta HTTP successiva e il server lo utilizza per verificare se l'utente si è autenticato.
Viene fatto un controllo sulla validità del token senza salvare informazioni sull'utente.

Questo approccio è particolarmente utile per le applicazioni multipiattaforma (web, desktop, mobile).

autenticazione con token

Vantaggi:

- Scalabilità: i token sono salvati localmente dal client, quindi il server non deve occuparsi di gestire gli oggetti di sessione.
- Multipiattaforma: lo stesso token può rappresentare lo stesso utente sia su un'applicazione web che mobile (l'ID di sessione non potrebbe essere condiviso da diversi *frontends*).

Svantaggi:

- Inefficienza: un token contiene tutte le informazioni sull'utente, quindi scambiare un token per ogni richiesta può incidere negativamente sulle prestazioni.

UNIMORE
UNIVERSITÀ DEGLI STUDI DI MODENA E REGGIO EMILIA

token e Django REST

Django REST Framework utilizza l'autenticazione con token, integrando `TokenAuthentication`. È un'implementazione molto scarna, per esempio non include il tempo di scadenza del token.

Attraverso applicazioni di terze parti si possono aggiungere i JSON Web Tokens (JWTs), che permettono di generare token univoci e tempo di scadenza. Uno di questi servizi e Auth0.

I JWTs possono anche essere crittografati, quindi utilizzabili da protocolli HTTP non sicuri.

django REST framework

UNIMORE
UNIVERSITÀ DEGLI STUDI DI MODENA E REGGIO EMILIA

impostare autenticazione

Nel file `settings.py` si aggiorna il dizionario `REST_FRAMEWORK`:

```
REST_FRAMEWORK = {
    ‘DEFAULT_PERMISSION_CLASSES’: [
        ‘rest_framework.permissions.IsAuthenticated’,
    ],
    ‘DEFAULT_AUTHENTICATION_CLASSES’: [
        ‘rest_framework.authentication.SessionAuthentication’,
        ‘rest_framework.authentication.TokenAuthentication’,
    ],
}
```

django REST framework

UNIMORE
UNIVERSITÀ DEGLI STUDI DI MODENA E REGGIO EMILIA

impostare autenticazione

`SessionAuthentication` si utilizza per le Browsable API (?)

`TokenAuthentication` serve per passare il token negli headers della richiesta HTTP.

È necessario aggiungere anche l'applicazione `rest_framework.authtoken` alla lista `INSTALLED_APPS` nel file `settings.py`.

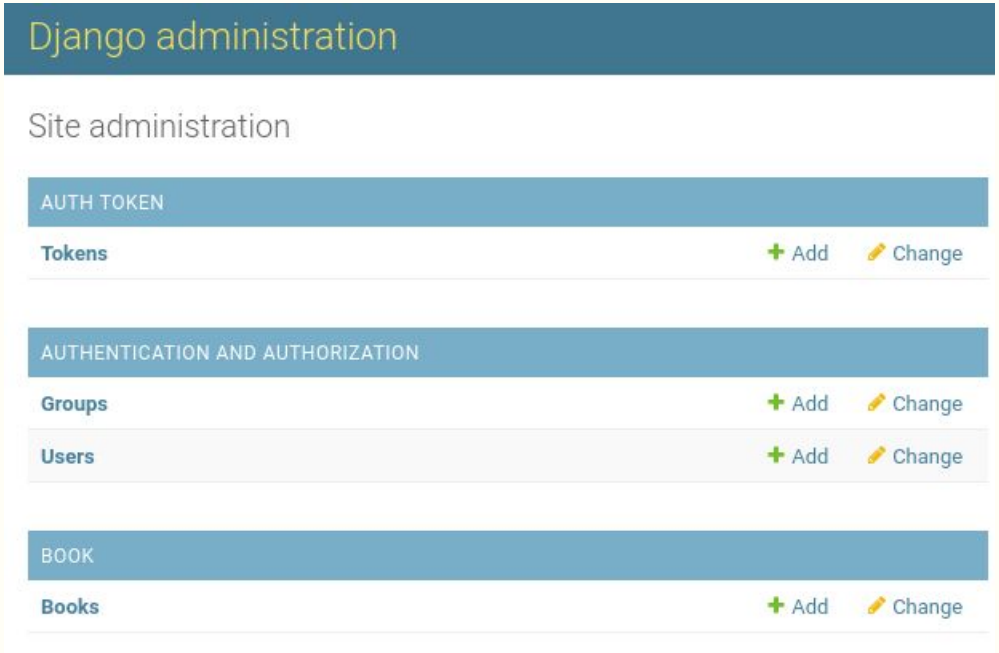
Avendo aggiunto una nuova applicazione, si sincronizza il database:

```
python manage.py migrate
```

django REST framework

token aggiunti al database

Nell'interfaccia di admin è ora presente il modello `Token` dell'applicazione `authtoken`:



Non sono presenti record di token perché gli utenti creati fino ad ora non hanno fatto uso di questo meccanismo per autenticarsi.

UNIMORE
UNIVERSITÀ DEGLI STUDI DI MODENA E REGGIO EMILIA

dj-rest-auth

Per l'autenticazione si utilizza l'applicazione dj-rest-auth.

```
pip install django-rest-auth
```

Nel file `settings.py` si aggiunge l'applicazione `dj_rest_auth` alla lista `INSTALLED_APPS`.

Per accedere agli endpoints di questa applicazione, è necessario aggiungere gli url di `dj_rest_auth` nel file `urls.py` del progetto:

```
urlpatterns = [
    …,
    path('api/v1/dj-rest-auth/', include('dj_rest_auth.urls')),
]
```

django REST framework

UNIMORE
UNIVERSITÀ DEGLI STUDI DI MODENA E REGGIO EMILIA

dj-rest-auth - login

localhost:8000/api/v1/dj-rest-auth/login/

Login

OPTIONS

Check the credentials and return the REST Token
if the credentials are valid and authenticated.
Calls Django Auth login method to register User ID
in Django session framework

Accept the following POST parameters: username, password
Return the REST Framework Token Object's key.

```
GET /api/v1/dj-rest-auth/login/
```

```
HTTP 405 Method Not Allowed
Allow: POST, OPTIONS
Content-Type: application/json
Vary: Accept

{
    "detail": "Method \"GET\" not allowed."
}
```

Raw data | HTML form

Username

Email

Password

POST

django REST framework

UNIMORE
UNIVERSITÀ DEGLI STUDI DI MODENA E REGGIO EMILIA

dj-rest-auth - logout

localhost:8000/api/v1/dj-rest-auth/logout/

Logout

OPTIONS GET

Calls Django logout method and delete the Token object assigned to the current User object.

Accepts/Returns nothing.

```
GET /api/v1/dj-rest-auth/logout/
```

```
HTTP 405 Method Not Allowed
Allow: GET, POST, HEAD, OPTIONS
Content-Type: application/json
Vary: Accept

{
    "detail": "Method \"GET\" not allowed."
}
```

Media type: application/json

Content:

POST

django REST framework

UNIMORE
UNIVERSITÀ DEGLI STUDI DI MODENA E REGGIO EMILIA

dj-rest-auth - reset password

localhost:8000/api/v1/dj-rest-auth/password/reset/

Password Reset

OPTIONS

Calls Django Auth PasswordResetForm save method.

Accepts the following POST parameters: email
Returns the success/fail message.

```
GET /api/v1/dj-rest-auth/password/reset/
```

```
HTTP 405 Method Not Allowed
Allow: POST, OPTIONS
Content-Type: application/json
Vary: Accept

{
    "detail": "Method \"GET\" not allowed."
}
```

Raw data | HTML form

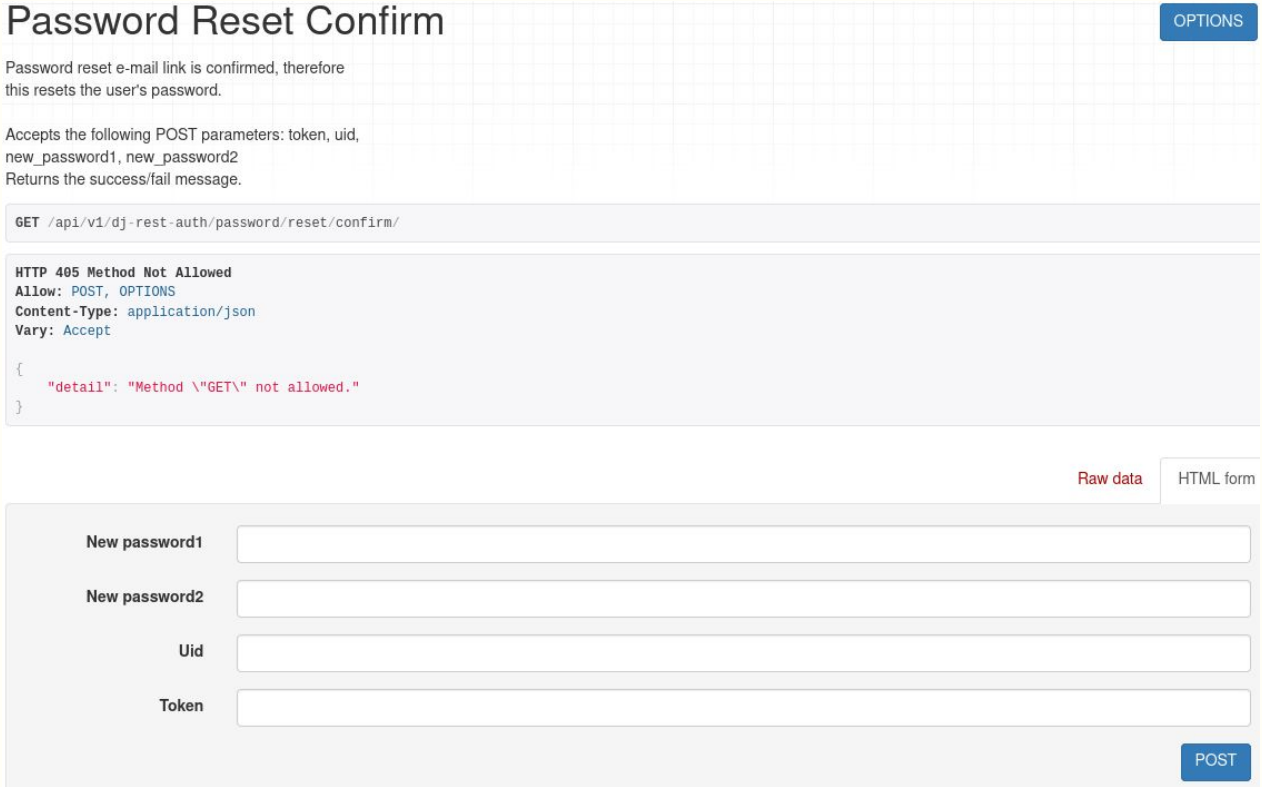
Email

POST

UNIMORE
UNIVERSITÀ DEGLI STUDI DI MODENA E REGGIO EMILIA

dj-rest-auth - confirm reset

localhost:8000/api/v1/dj-rest-auth/password/reset/confirm/



django REST framework

UNIMORE
UNIVERSITÀ DEGLI STUDI DI MODENA E REGGIO EMILIA

registrazione utente

Per gestire la registrazione degli utenti si usa l'applicazione `django-allauth`:

```
pip install django-allauth
```

Nel file `settings.py del progetto`, aggiungere alla lista `INSTALLED_APPS`:

```
INSTALLED_APPS = [
    …,
    ‘django.contrib.sites’,
    ‘allauth’,
    ‘allauth.account’,
    ‘allauth.socialaccount’,
    ‘dj_rest_auth.registration’,
]
```

django REST framework

registrazione utente

Nel file `settings.py` del progetto aggiungere anche:

```
EMAIL_BACKEND = ‘django.core.mail.backends.console.EmailBackend’
SITE_ID = 1
```

`EMAIL_BACKEND` è configurata in modo che le email vengano mostrate nella console. In questo caso non c’è bisogno di configurare anche un server per le email.

Django permette di hostare più siti utilizzando lo stesso progetto. In questo caso viene hostato un solo sito, con IP 127.0.0.1 (localhost), quindi si configura `SITE_ID = 1`. ???

UNIMORE
UNIVERSITÀ DEGLI STUDI DI MODENA E REGGIO EMILIA

registrazione utente

Avendo aggiunto una nuova applicazione che interagisce col database, eseguire il comando:

```
pyhton manage.py migrate
```

Nel file `urls.py` del progetto, aggiungere:

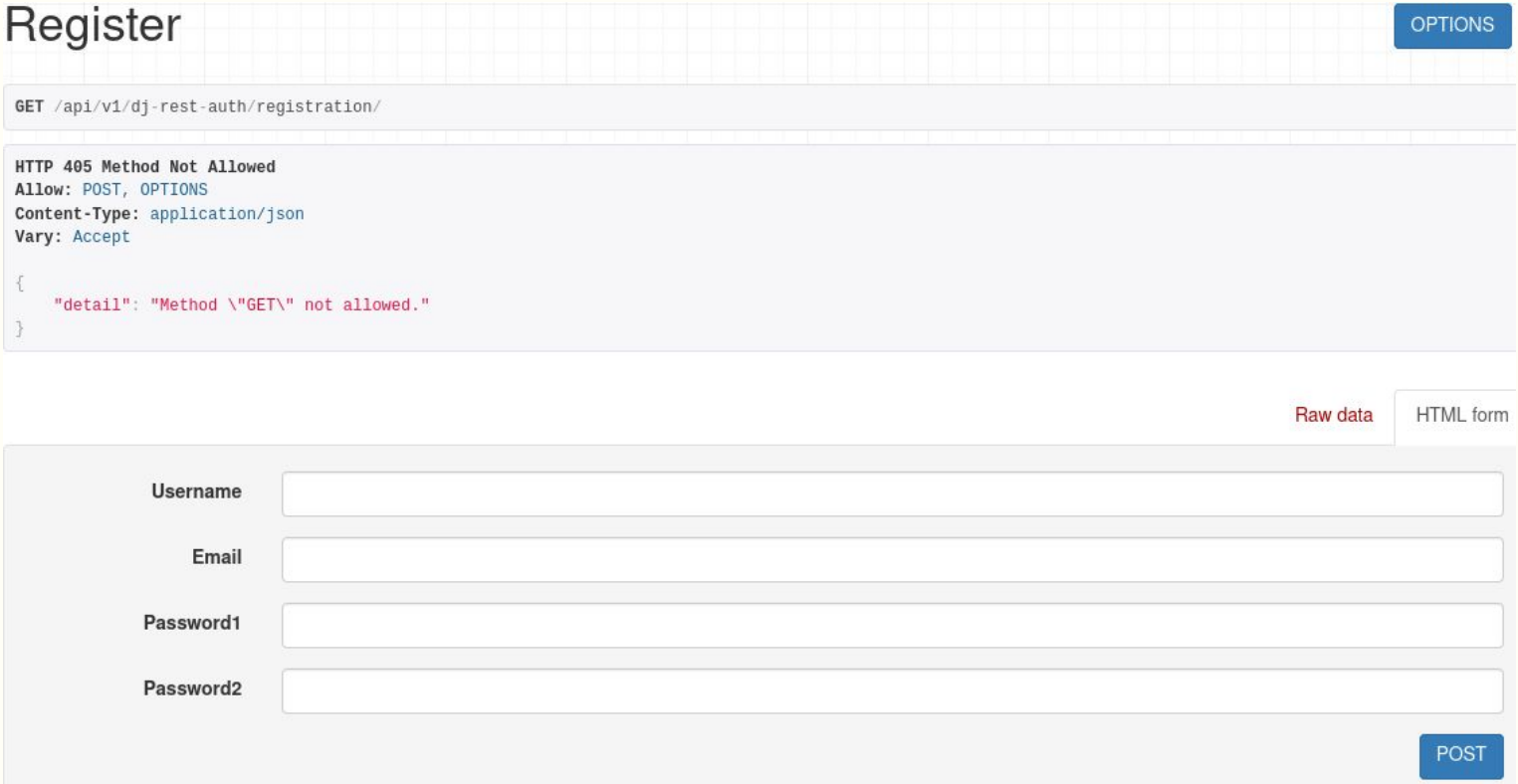
```
urlpatterns = [
    …,
    path(‘api/v1/dj-rest-auth/registration/’,
        include(‘dj_rest_auth.registration.urls’)),
]
```

django REST framework

UNIMORE
UNIVERSITÀ DEGLI STUDI DI MODENA E REGGIO EMILIA

dj-rest-auth - registration

localhost:8000/api/v1/dj-rest-auth/registration/



django REST framework

UNIMORE
UNIVERSITÀ DEGLI STUDI DI MODENA E REGGIO EMILIA

conferma email (console)

```
Content-Type: text/plain; charset="utf-8"
MIME-Version: 1.0
Content-Transfer-Encoding: 7bit
Subject: [example.com] Please Confirm Your E-mail Address
From: webmaster@localhost
To: myemailaddress@gmail.com
Date: Mon, 26 Oct 2020 10:38:52 -0000
Message-ID: <160370873281.41777.16338703187892489333@debian>

Hello from example.com!

You're receiving this e-mail because user myusername has given your e-mail address to register an
account on example.com.

To confirm this is correct, go to
http://localhost:8000/api/v1/dj-rest-auth/registration/account-confirm-email/MQ:1kWzts:nbTRGWh1BITFO
0_z4hWVejVGpZBB9F6dNcWOJPu5meA/

Thank you for using example.com!
example.com
```

django REST framework

UNIMORE
UNIVERSITÀ DEGLI STUDI DI MODENA E REGGIO EMILIA

dj-rest-auth - token

Navigando l'interfaccia di admin ora è possibile vedere nella tabella Token un record, corrispondente all'utente appena creato.

Questo token viene salvato nel localStorage del client o come cookie. Le future richieste conterranno questo token nell'header per autenticare l'utente.

django REST framework

login con app di terze parti

Django REST framework permette di autenticarsi anche con servizi di terze parti, come Google, Facebook, Twitter ecc.

Grazie per l'attenzione

francesco.faenza@unimore.it