

کار با form ها

در این بخش از آموزش Django روش کار با form های HTML در Django را بررسی خواهیم کرد و به طور ویژه به بررسی ساده ترین روش ایجاد، به روز رسانی و حذف instance های مدل با استفاده از form ها میپردازیم. همچنین به عنوان بخشی از آموزش، وب سایت LocalLibrary را گسترش داده و با استفاده از form ها (به جای استفاده از admin application) به کتابداران اجازه می‌دهیم مهلت امانت کتاب ها را تجدید کرده، نویسنده جدید ایجاد کرده و یا نویسندگان موجود را حذف و به روز رسانی کنند.

پیش نیاز ها: تمام بخش های قبلی آموزش، شامل احراز هویت کاربر و مجوز ها، را مطالعه کنید.

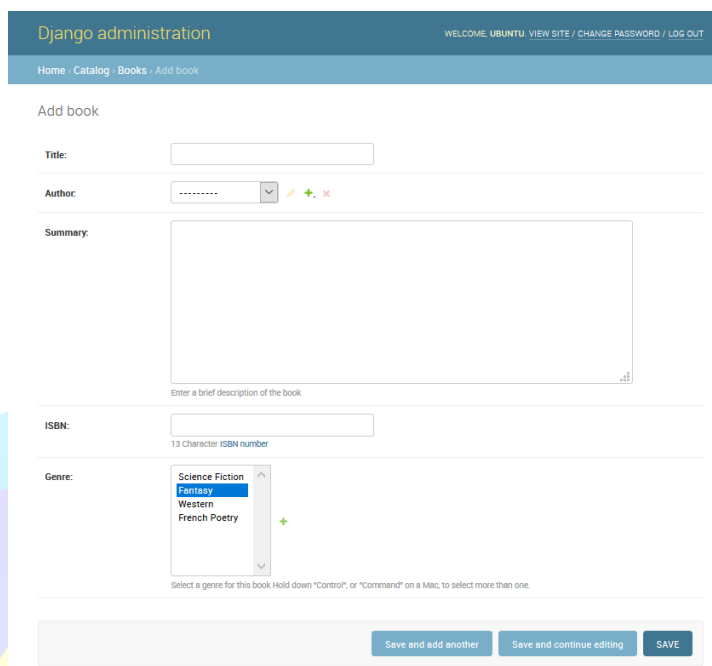
اهداف: فهم چگونگی نوشتن form به منظور دریافت اطلاعات از کاربران و به روز رسانی دیتابیس. با استفاده از generic class based view ها می‌توانیم، به سادگی، فرم های مورد نیاز در یک مدل را بسازیم.

نگاه کلی

یک HTML Form، شامل مجموعه ای از فیلد ها و یا widget های موجود در یک وب سایت است که برای دریافت اطلاعات از کاربر و ارسال آن به سرور استفاده میشود. form ها مکانیزم های انعطاف پذیری هستند که به وسیله ی آنها میتوان انواع داده ها را از طریق text box، checkbox، radio buttons، انتخاب کننده های تاریخ و ... از کاربر دریافت کرد. form ها همچنین، به علت قابلیت ارسال داده ها در POST request با استفاده از محافظت cross-site request forgery، از امنیت نسبتا بالایی برخوردارند. با اینکه تا به حال در این آموزش ها هیچ form ای ایجاد نکرده ایم اما در Django Admin site با آنها روبرو شده ایم. برای مثال، شکل زیر نشان دهنده یک form، متشکل از تعدادی لیست انتخابی و ویرایشگر

متن، در حال ویرایش مدل های Book میباشد.

آموزش Django



کار با فرم ها میتواند قدری پیچیده باشد! برنامه نویسان باید برای یک فرم کد های لازم HTML را نوشته، داده وارد شده به سرور (و گاهی جستجو گر) را تایید و پاکسازی کنند.

احتمالا به کاربر پیام خطایی ارسال کرده و آنها را از وجود فیلد های پر نشده و یا با داده نادرست پر شده مطلع کنند. داده صحیح را پس از دریافت مدیریت کرده و در نهایت به شکلی به کاربر اطلاع دهند که عملیات با موفقیت انجام شده است.

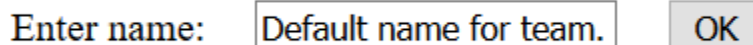
Django ، با فراهم کردن یک framework ، به شما امکان تعریف form مورد نظرتان و فیلد های آن را داده و از این اشیا برای ایجاد کد HTML مورد نیاز form و تایید مدیریت ارتباطات با کاربران استفاده میکند.

در این آموزش، چند روش برای کار با فرم ها و به طور خاص، تاثیر به سزای generic editing view ها بر کاهش عملیات مورد نیاز برای ایجاد فرم ها و مدیریت مدل ها را بررسی خواهیم کرد.

همچنین به عنوان بخشی از آموزش، وب سایت LocalLibrary را گسترش داده و فرم های جدیدی ایجاد میکنیم تا کتابداران بتوانند مهلت امانت کتاب های کتابخانه را تجدید کرده و با استفاده از صفحات جدید، نویسندگان و کتاب های تازه ایجاد شده و یا نویسندگان و کتاب های موجود در لیست را حذف و به روز رسانی کنند.

HTML Form ها

یک HTML form ساده را، که دارای فیلد متنی به منظور وارد کردن نام یک "تیم" و یک label مرتبط است، در نظر بگیرید:



این فرم HTML به کمک مجموعه ای از المان های درون `<form>...</form>` تعریف شده و حداقل دارای یک المان `input` به شکل `type="submit"` میباشد:

```
<form action="/team_name_url/" method="post">

    <label for="team_name">Enter name: </label>

    <input id="team_name" type="text" name="name_field"
value="Default name for team.">

    <input type="submit" value="OK">

</form>
```

با اینکه در این مثال تنها یک فیلد متنی موجود است، اما میتوان به هر تعداد دلخواه فیلد ورودی و برچسب های مرتبط برای آنها ایجاد کرد. فیلد `type`، نشان دهنده نوع widget نمایش داده شده است.

`name` و `id` فیلد برای تشخیص فیلد در JavaScript/CSS/HTML استفاده شده و `value`، مقدار اولیه فیلد را در اولین نمایش مشخص میکند. برچسب فیلد ورودی نام تیم با استفاده از `label tag` ("Enter name" در شکل بالا) و یک فیلد `for` حاوی مقدار `id` ورودی مرتبط، ایجاد شده است.

ورودی `submit` (به شکل پیش فرض) به عنوان یک `button` نمایش داده میشود که کاربر میتواند بر آن کلیک کرده و المان های ورودی دیگر (در این مثال تنها `team_name`) را به سرور ارسال کند. صفات

آموزش Django

(attribute) form ها، متد HTML استفاده شده برای ارسال و مقصد داده در سرور، را تعریف میکنند (action) :

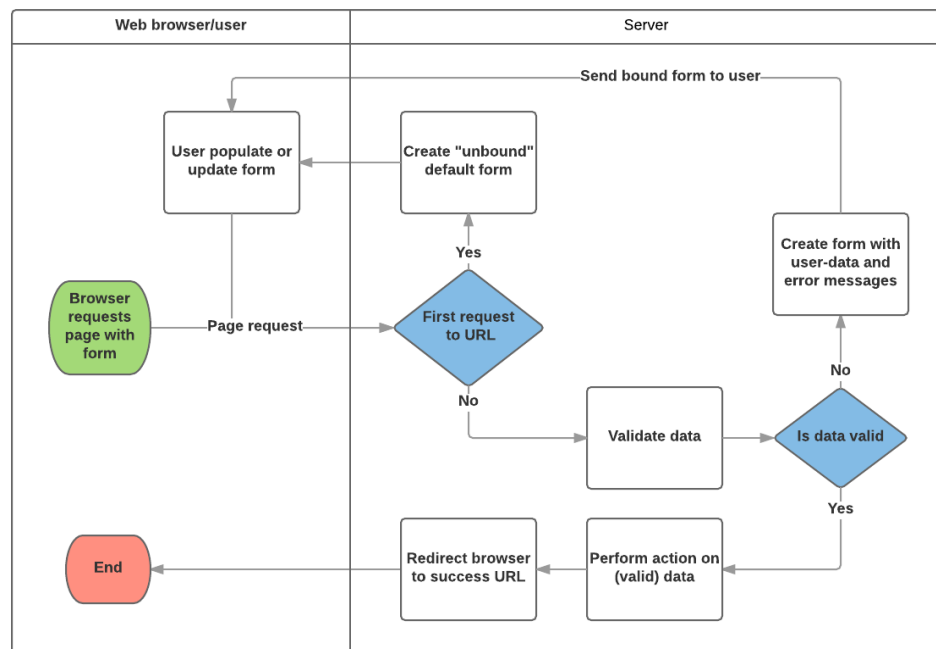
- action – منبع/URL ای که داده ها، پس از submit فرم ، برای پردازش به آن ارسال میشوند.
 - method – متد HTTP استفاده شده برای ارسال داده ها : post یا get
 - متد POST، برای داده هایی که منجر به تغییری در دیتابیس سرور میشوند، استفاده میشود زیرا این متد مقاومت بیشتری در برابر حملات cross-site forgery request دارد.
 - متد GET، تنها برای فرم هایی که تغییری در داده های کاربر ایجاد نمیکند استفاده شود (مانند فرم های جستجو). توصیه میشود در مواقعی که نیاز به bookmark و یا اشتراک گذاری یک URL دارید از این متد استفاده کنید.
- نقش سرور، ابتدا render کردن وضعیت ابتدایی فرم برای بررسی فیلد های خالی و یا پر شده با مقادیر اولیه است.
- پس از فشردن شدن دکمه submit توسط کاربر، سرور داده های فرم و مقادیر جستجوگر را دریافت کرده و باید این اطلاعات را تایید کند.
- اگر فرم حاوی اطلاعات نامعتبر باشد، سرور دوباره فرم را به کاربر نمایش خواهد داد، با این تفاوت که مقادیر فیلد های معتبر حفظ شده و برای فیلد های نامعتبر، توضیحی نمایش داده خواهد شد.
- پس از اینکه سرور درخواستی با تمامی مقادیر معتبر دریافت کرد، میتواند عملیات مناسب را انجام داده (مثلا داده ها را ذخیره کند، نتیجه جستجو ای را نمایش دهد، فایلی را آپلود کند و ...) و کاربر را مطلع کند.
- انجام صحیح این مراحل (ساخت HTML، تایید داده های دریافتی، نمایش دوباره داده های دریافتی به همراه گزارش error های موجود،انجام عملیات لازم بر داده های تایید شده) کار نسبتا دشواری است. Django، با به عهده گرفتن بخش های سنگین و کد های تکراری، انجام این کار ها را برایمان بسیار ساده میکند!

فرآیند Django در form handling

form handling در Django از تکنیک هایی مشابه آنچه در بخش های قبل آموختیم (برای نمایش اطلاعات مربوط به مدل هایمان) استفاده میکند:

ابتدا **view** یک درخواست دریافت میکند، عملیات مورد نظر شامل خواندن داده ها از مدل را انجام داده و سپس یک صفحه HTML ایجاد کرده و ارسال میکند (با استفاده از **template** ، که از طریق آن **context** حاوی داده هایی که باید نمایش داده شوند، را پاس میکنیم). نیاز به پردازش داده هایی که توسط کاربر به سرور ارسال شده و نمایش دوباره صفحه در صورت وجود خطا، بخش پیچیده این عملیات است.

فلوچارت پردازش form handling در Django در شکل زیر نمایش داده شده است. این عملیات با ارسال درخواست نمایش صفحه حاوی فرم (بخش سبز رنگ) آغاز میشود.



بر اساس دیاگرام بالا، موارد زیر اصلی ترین بخش های form handling در Django هستند:

1. نمایش form پیش فرض با اولین درخواست کاربر

آموزش Django

- فیلد های این فرم ممکن است خالی (مثلا اگر در حال ایجاد record جدیدی هستیم) و یا حاوی مقادیر اولیه (مثلا اگر در حال تغییر یک record هستیم و یا مقادیر اولیه اطلاعات مفیدی به کاربر نمایش میدهند) باشند.
- فرم، unbound (بدون محدودیت) در نظر گرفته میشود ، زیرا که با داده های وارد شده از طرف کاربران ارتباطی ندارد (با این حال ممکن است مقادیر اولیه ای در آن موجود باشد).
- 2. دریافت داده ها از یک درخواست submit و bind کردن آنها به form
 - bind کردن داده ها به فرم، به معنای در دسترس بودن داده های وارد شده توسط کاربر و خطا های آن در زمان نمایش دوباره ی فرم است.
- 3. پاکسازی و تایید داده ها
 - Cleaning data. داده های ورودی را پاکسازی کرده (مثلا حذف کاراکتر های نا معتبر که ممکن است به منظور ارسال محتوای مخرب به سرور استفاده شده باشند) و آنها را به type های پایتون تبدیل میکند.
 - اعتبار سنجی (validation) تناسب داده ها با فیلد را بررسی میکند (مثلا بیش از حد طولانی و یا کوتاه بودن یک بازه زمانی و ...)
- 4. در صورت وجود داده نا معتبر، نمایش دوباره فرم همراه با مقادیر وارد شده توسط کاربر و خطاهای مربوط به فیلد های مشکل دار.
- 5. در صورت معتبر بودن همه داده ها، عملیات درخواستی (مثلا ذخیره داد ها، ارسال یک ایمیل، پاسخ های موجود برای نتیجه یک جستجو، آپلود یک فایل و ...) انجام می شود.
- 6. هدایت کاربر به صفحه ای جدید، در صورت اتمام عملیات.

Django ابزار و روش هایی برای کمک به انجام مراحل بالا فراهم کرده است که اصلی ترین آنها کلاس Form میباشد. این کلاس تولید کد HTML و تایید/پاکسازی داده ها را برای ما آسان میکند. در بخش بعد روش کار فرم ها را، با استفاده از مثال تجدید کتاب ها توسط کتابداران، بررسی میکنیم.

توجه

فهم روش کار با Form ها برای بخش های بعدی، که بررسی کلاس های سطح بالا Django میباشد، نیز مفید است.

فرم تجدید کتاب (Renew-book form) با استفاده از یک Form و یک function view

در این بخش صفحه ای ایجاد میکنیم که به کتابداران اجازه میدهد کتاب های امانت گرفته شده را تجدید کنند. به این منظور برای کتابداران یک فرم ایجاد میکنیم تا بتوانند تاریخی در آن وارد کنند. مقدار اولیه این تاریخ را سه هفته پس از تاریخ حال حاضر قرار داده (که بازه معمول امانت کتاب است) و با کمک validation از ورود تاریخی در آینده دور و یا گذشته، توسط کتابداران، جلوگیری میکنیم. پس از ورود مقدار معتبر، آن را در رکورد موجود فیلد `BookInstance.due_back` قرار میدهیم.

در این مثال از یک کلاس Form و یک function-based view استفاده میشود. در بخش بعد، روش کار فرم ها و تغییراتی که باید بر پروژه LocalLibrary ایجاد کنیم، را بررسی میکنیم.

Form

کلاس Form، اصلی ترین بخش سیستم form handling در Django میباشد. این کلاس فیلد های فرم، layout های آنها، نمایش widget ها، برچسب ها، مقادیر اولیه، مقادیر معتبر و (پس از تایید اعتبار) خطاهای مرتبط با فیلد های نامعتبر، را مشخص میکند.

همچنین متد هایی برای render خودش در template، با استفاده از فرمت های از پیش تعریف شده (جدول ها، لیست ها، ...) و یا دریافت مقدار هر المان (با فعال سازی fine-grained manual rendering) فراهم میکند.

تعریف یک Form

تعریف Form بسیار شبیه تعریف یک Model بوده و field type های یکسانی دارد (برخی از پارامتر های آنها نیز یکسان هستند) که کاملاً قابل درک است زیرا در هر دو مورد فیلد ها باید نوع داده مناسب و معتبری داشته و توضیحاتی برای documentation/display داشته باشند.

داده های Form در فایل forms.py اپلیکیشن، در دایرکتوری application، ذخیره میشوند. فایل **locallibrary/catalog/forms.py** را ایجاد و باز کنید.

برای ایجاد یک Form، کتابخانه form را import کرده و از کلاس Form مشتق میگیریم، سپس فیلد های form را تعریف میکنیم. یک کلاس فرم خیلی ساده برای فرم تجدید کتاب در بخش زیر نمایش داده شده است، آن را به فایل جدیدی که ایجاد کرده اید بیفزایید :

```
from django import forms

class RenewBookForm(forms.Form):

    renewal_date = forms.DateField(help_text="Enter a date
between now and 4 weeks (default 3).")
```

آموزشگاه تحلیل داده

فیلد های فرم

در فرم بالا، یک **DateField** برای دریافت تاریخ تجدید که با یک مقدار blank در HTML، render میشود، برچسب پیش فرض "Renewal date:" و متنی کمکی

("Enter a date between now and 4 week(default 3)") موجود است.

از آنجایی که آرگومان های optional دیگر در فیلد مشخص نشده اند، تاریخ با [input formats](#) (MM/DD/YYYY (02/26/2016), YYYY-MM-DD (2016-11-06))،

آموزش Django

(MM/DD/YY (10/25/16 پذیرفته شده و با استفاده از widget پیش فرض ([DateInput](#)) render میشود.

form field ها انواع بسیاری دارند که با توجه به تشابه شان به فیلد های مدل کلاس ها قابل شناسایی اند :

[CharField](#), [ChoiceField](#), [TypedChoiceField](#), [DateField](#), [DateTimeField](#), [DecimalField](#), [DurationField](#), [EmailField](#), [FileField](#), [FilePathField](#), [FloatField](#), [ImageField](#), [IntegerField](#), [GenericIPAddressField](#), [MultipleChoiceField](#), [TypedMultipleChoiceField](#), [NullBooleanField](#), [RegexField](#), [SlugField](#), [TimeField](#), [URLField](#), [UUIDField](#), [ComboField](#), [MultiValueField](#), [SplitDateTimeField](#), [ModelMultipleChoiceField](#), [ModelChoiceField](#)

آرگومان های موجود در لیست زیر، بین اکثر فیلد ها مشترک اند (و دارای مقادیر پیش فرض مناسب هستند):

- [required](#) - اگر True باشد، فیلد نمیتواند خالی بوده و یا مقدار None بپذیرد. فیلد ها به شکل پیش فرض `required` هستند، پس اگر مایل به خالی گذاشتن فیلد هستید باید `required=False` قرار دهید.
- [label](#) - برچسب استفاده شده در زمان render فیلد در HTML. اگر `label` مشخص نشده باشد، Django با استفاده از نام فیلد (تغییر حرف اول به حرف بزرگ و قرار دادن فاصله به جای زیر خط) یک برچسب برای فیلد خواهد ساخت (مثلا `Renewal date`).
- [label suffix](#) - به شکل پیش فرض پس از `label` یک دونقطه قرار میگیرد (مثلا `Renewal date:`). این آرگومان به شما امکان استفاده از یک پسوند محتوی کاراکتر (یا کاراکتر های) نیز دیگری میدهد.
- [initial](#) - مقدار اولیه فیلد در زمان نمایش فرم.
- [widget](#) - `display widget` مورد استفاده.
- [help_text](#) - (همانطور که در مثال بالا مشاهده کردید) متنی که برای ارائه توضیحات بیشتر درباره فیلد نمایش داده میشود.
- [error_messages](#) - لیستی از خطا های موجود در فیلد. میتوانید این بخش را با پیام دیگری override کنید.

Django آموزش

- [validator](#) – لیستی از توابعی که پس از تایید اعتبار فیلد فراخوانی میشوند.
- [localize](#) – فعال سازی localization داده های ورودی فرم
- [disabled](#) – اگر True باشد، فیلد نمایش داده میشود، اما مقدار آن ویرایش نمیشود. مقدار پیش فرض این آرگومان False است.

Validation

Django امکان تایید داده ها را در بخش های مختلفی برای شما فراهم میکند. ساده ترین روش تایید یک فیلد، `override` کردن متد `clean_<fieldname>()` برای فیلد مورد نظر است.

به عنوان مثال، برای بررسی `renewal_date` و اطمینان از اینکه تاریخ وارد شده بین الان و چهار هفته آینده است، از `clean_renewal_date()` به شکل زیر استفاده میکنیم.

فایل `forms.py` خود را به شکل زیر به روز رسانی کنید:

```
import datetime

from django import forms

from django.core.exceptions import ValidationError

from django.utils.translation import ugettext_lazy as _

class RenewBookForm(forms.Form):

    renewal_date = forms.DateField(help_text="Enter a date
    between now and 4 weeks (default 3).")
```

```
def clean_renewal_date(self):  
  
    data = self.cleaned_data['renewal_date']  
  
    # Check if a date is not in the past.  
  
    if data < datetime.date.today():  
  
        raise ValidationError(_('Invalid date - renewal  
in past'))  
  
    # Check if a date is in the allowed range (+4 weeks  
from today).  
  
    if data > datetime.date.today() +  
datetime.timedelta(weeks=4):  
  
        raise ValidationError(_('Invalid date - renewal  
more than 4 weeks ahead'))  
  
    # Remember to always return the cleaned data.  
  
    return data
```

در این بخش لازم است به دو مورد مهم توجه کنید.

مورد اول این است که ما داده های خود را با استفاده از `self.cleaned_data['renewal_date']` دریافت کرده و پس از اتمام تابع، صرف نظر از اینکه تغییری بر آنها انجام شده باشد یا نه، داده را باز میگردانیم.

آموزش Django

این گام، با استفاده از validator پیش فرض، داده های ما را از ورودی های نا امن احتمالی پاکسازی کرده و آنها را به type های standard و صحیح تبدیل میکند (در این مثال شی `datetime.datetime` پایتون).

مورد مهم بعدی این است که اگر مقادیر ورودی خارج از محدوده مشخص شده ما باشد، یک `ValidationError` ایجاد شده و متنی (مشخص شده توسط ما) که نشان دهنده ورود مقدار نادرست است نمایش میدهد. مثال بالا این متن را در قالب یک `Django's translation functions`، `ugettext_lazy()`، (`import` شده با `_()`) قرار میدهد، که ابزار مفیدی برای ترجمه سایت نیز میباشد.

توجه

متد ها و مثال های دیگری نیز برای اعتبار سنجی (`validate`) فرم ها در [Form and field validation](#) موجود است. برای مثال، اگر چندین فیلد وابسته به هم داشته باشید، میتوانید تابع [Form.clean\(\)](#) را `override` کرده و دوباره `ValidationError` ایجاد کنید.

در این مثال به مورد دیگری از `form` ها نیاز نداریم !

URL configuration

پیش از اینکه `view` خود را بسازیم، ابتدا باید یک `URL configuration` برای صفحه `renew_book` ایجاد کنیم. تنظیمات زیر را به انتهای `locallibrary/catalog/urls.py` اضافه کنید:

```
urlpatterns += [

    path('book/<uuid:pk>/renew/', views.renew_book_librarian,
name='renew-book-librarian'),

]
```

این `URL configuration`، `URL` هایی با فرمت

`/catalog/book/<bookinstance_id>/renew/` را به تابعی با

نام `renew_book_librarian()` در `views.py` هدایت کرده و `BookInstance id` را در پارامتری به نام `pk` ارسال میکند. این `pattern` تنها در صورتی که فرمت `pk`, `uuid` باشد تطبیق پیدا میکند.

توجه

از آنجا که کنترل کامل بر تابع `view` داریم، میتوانیم به جای نام `pk` برای داده ی `URL` دریافت شده، هر نام دلخواهی انتخاب کنیم (از `generic detail view class` که پارامترهایی با نام مشخص دریافت میکند استفاده نمیکنیم). ولی با توجه به اینکه `pk` مخفف "`primary key`" میباشد، انتخاب مناسبی است.

View

همانطور که در [Django form handling process](#) بالا بحث شد، `view` باید در اولین فراخوانی فرم پیش فرض را `render` کند. سپس اگر داده نا معتبری یافته شد، آن را به همراه پیام های خطا `re-render` کرده و اگر داده ها معتبر بود، داده ها را پردازش کرده و به صفحه جدیدی هدایت شود. برای اجرای این عملیات، `view` باید تشخیص دهد که برای اولین بار فراخوانی شده تا فرم پیش فرض را `render` کند و یا اینکه به داده معتبر دسترسی دارد.

در فرم هایی که از درخواست `POST` برای `submit` اطلاعات به سرور استفاده میکنند، متداول ترین الگو، بررسی درخواست `POST` توسط `view` میباشد (`if request.method == 'POST':`) تا بتواند درخواست های اعتبار سنجی فرم را شناسایی کرده و با استفاده از `GET` (و یک شرط `else`) درخواست های ایجاد فرم را بیابد.

اگر بخواهید با استفاده از `GET` داده های خود را `submit` کنید، روش معمول برای شناسایی اولین درخواست و یا درخواست های زیر مجموعه، استناد به `view` و خواندن داده های فرم میباشد (مثلا خواندن داده ای `hidden` در فرم).

فرآیند تجدید مهلت کتاب بر دیتابیس نوشته خواهد شد، پس بر طبق قرارداد از روش درخواست `POST` استفاده میکنیم. قطعه کد زیر نشان دهنده ی یک الگو (بسیار استاندارد) برای توابع `view` این چنینی است:

```
import datetime

from django.shortcuts import render, get_object_or_404
from django.http import HttpResponseRedirect
from django.urls import reverse

from catalog.forms import RenewBookForm

def renew_book_librarian(request, pk):
    book_instance = get_object_or_404(BookInstance, pk=pk)

    # If this is a POST request then process the Form data
    if request.method == 'POST':

        # Create a form instance and populate it with data
        # from the request (binding):
        form = RenewBookForm(request.POST)

        # Check if the form is valid:
```

```
if form.is_valid():

    # process the data in form.cleaned_data as
    required (here we just write it to the model due_back field)

    book_instance.due_back =
form.cleaned_data['renewal_date']

    book_instance.save()

    # redirect to a new URL:

    return HttpResponseRedirect(reverse('all-
borrowed')) )

# If this is a GET (or any other method) create the
default form.

else:

    proposed_renewal_date = datetime.date.today() +
datetime.timedelta(weeks=3)

    form = RenewBookForm(initial={'renewal_date':
proposed_renewal_date})

    context = {

        'form': form,

        'book_instance': book_instance,
```

```
}

return render(request,
'catalog/book_renew_librarian.html', context)
```

ابتدا، فرم خود (RenewBookForm) و تعداد دیگری از اشیا و متد های مفید برای بدنه ی تابع view را import میکنیم:

- [get_object_or_404\(\)](#): یک شی مشخص را ، بر اساس مقدار کلید اصلی اش، از یک مدل برگردانده و در صورت عدم وجود رکورد، یک `Http404 exception` (یافت نشد – `not found`) ایجاد میکند.
- [HttpResponseRedirect](#) : ایجاد یک تغییر مسیر (`redirect`) به یک URL خاص (`HTTP status code 302`).
- [reverse\(\)](#) : ایجاد یک URL از نام یک `URL configuration` و تعیین آرگومان هایش. این متد معادل `url tag` در پایتون است که در `template` هایمان استفاده کردیم.
- [datetime](#) : یک کتابخانه پایتون برای تغییر تاریخ و زمان.

در این `view`، ابتدا از آرگومان `pk` در `(get_object_or_404)` استفاده کرده و `BookInstance` حال حاضر را دریافت میکنیم (اگر موجود نباشد، `view` بلافاصله `exit` شده و خطا “not found” نمایش میدهد).

اگر درخواست `POST` نباشد (توسط عبارت `else` انجام شود)، با پاس دادن یک مقدار اولیه برای فیلد `renewal_date` (همانطور که در بخش زیر نمایش داده شده، ۳ هفته بعد از تاریخ همانروز)، یک فرم پیش فرض ایجاد میکنیم.

```
book_instance = get_object_or_404(BookInstance, pk=pk)
```



```
# If this is a GET (or any other method) create the
default form

else:

    proposed_renewal_date = datetime.date.today() +
datetime.timedelta(weeks=3)

    form = RenewBookForm(initial={'renewal_date':
proposed_renewal_date})

    context = {

        'form': form,

        'book_instance': book_instance,

    }

    return render(request,
'catalog/book_renew_librarian.html', context)
```

پس از ساخت فرم، با فراخوانی `render()`، که `template` و `context` محتوی فرم را مشخص میکند، صفحه HTML را ایجاد میکنیم. در این حالت `context` حاوی `BookInstance` نیز خواهد بود و برای کسب اطلاعاتی درباره کتابی که در حال تجدید مهلت آن هستیم نیز استفاده میشود.

اگر درخواست POST باشد، شی `form` را ایجاد کرده و داده های موجود در `request` را در آن قرار میدهیم. این فرآینده `binding` نام داشته و امکان معتبر سازی فرم را برایمان فراهم میکند. سپس با اجرای تمام کد های `validation` بر تمام فیلدها، شامل کد های `generic` برای اطمینان از معتبر بودن تاریخ و تابع `clean_renewal_date()` برای فرم مشخص شده به منظور اطمینان از بازه صحیح تاریخ، اعتبار فرم را بررسی میکنیم.

```
book_instance = get_object_or_404(BookInstance, pk=pk)

# If this is a POST request then process the Form data

if request.method == 'POST':

    # Create a form instance and populate it with data
    from the request (binding):

    form = RenewBookForm(request.POST)

    # Check if the form is valid:

    if form.is_valid():

        # process the data in form.cleaned_data as
        required (here we just write it to the model due_back field)

        book_instance.due_back =
form.cleaned_data['renewal_date']

        book_instance.save()

    # redirect to a new URL:

    return HttpResponseRedirect(reverse('all-
borrowed')) )
```

```
context = {  
    'form': form,  
    'book_instance': book_instance,  
}  
  
return render(request,  
               'catalog/book_renew_librarian.html', context)
```

اگر فرم معتبر نباشد، دوباره `render()` را فراخوانی میکنیم. اما این بار مقدار پاس داده شده توسط `context` شامل پیام های خطا نیز خواهد بود.

اگر فرم معتبر باشد، میتوانیم از طریق صفت `form.cleaned_data` به داده ها دسترسی پیدا کنیم (مثلا `data = form.cleaned_data['renewal_date']`). در این بخش تنها داده ها را در مقدار `due_back` مرتبط به شی `BookInstance` ذخیره میکنیم.

هشدار

آموزشگاه تحلیل داده

با اینکه میتوانید مستقیماً از طریق درخواست (مثلا `request.POST['renewal_date']` یا `request.GET['renewal_date']` در صورت استفاده از `GET request`) به داده ها دسترسی پیدا کنید، اما انجام این کار توصیه نمیشود. داده ها پس از پاکسازی تایید شده و به `python-friendly type` تبدیل میشوند.

گام آخر در بخش مدیریت فرم `view`، هدایت به صفحه ی دیگر (معمولاً صفحه "success") میباشد. در اینجا ما با استفاده از `HttpResponseRedirect` و `reverse()` به `view` با نام `'all'`

آموزش Django

'borrowed' (احراز هویت کاربر و مجوز ها ، ایجاد شده است) تغییر مسیر خواهیم داد. اگر این صفحه را ایجاد نکرده اید میتوانید از '/' URL استفاده کرده و کاربر را به home page هدایت کنید.

حالا تمامی بخش های مربوط به form handling انجام شده است، اما هنوز دسترسی view تجدید کتاب ها را به کتابداران وارد شده به سایت محدود نکرده ایم. با استفاده از @login_required از کاربر میخواهیم وارد سایت شده و از دکوراتور تابع @permission_required به همراه can_mark_returned برای صدور مجوز استفاده میکنیم (دکوراتور ها به ترتیب اجرا میشوند). توجه کنید که درواقع باید مجوز جدیدی در تنظیمات BookInstance ("can_renew") ایجاد میکردیم، اما برای سادگی کار از یک مجوز موجود استفاده میکنیم.

view نهایی به شکل زیر خواهد بود. این بخش را در انتهای locallibrary/catalog/views.py کپی کنید:

```
import datetime

from django.contrib.auth.decorators import login_required,
permission_required

from django.shortcuts import get_object_or_404

from django.http import HttpResponseRedirect

from django.urls import reverse

from catalog.forms import RenewBookForm

@login_required
```

```
@permission_required('catalog.can_mark_returned',
raise_exception=True)

def renew_book_librarian(request, pk):

    """View function for renewing a specific BookInstance by
    librarian."""

    book_instance = get_object_or_404(BookInstance, pk=pk)

    # If this is a POST request then process the Form data

    if request.method == 'POST':

        # Create a form instance and populate it with data
        from the request (binding):

        form = RenewBookForm(request.POST)

        # Check if the form is valid:

        if form.is_valid():

            # process the data in form.cleaned_data as
            required (here we just write it to the model due_back field)

            book_instance.due_back =
            form.cleaned_data['renewal_date']

            book_instance.save()
```

```
        # redirect to a new URL:

        return HttpResponseRedirect(reverse('all-
borrowed') )

    # If this is a GET (or any other method) create the
    default form.

    else:

        proposed_renewal_date = datetime.date.today() +
datetime.timedelta(weeks=3)

        form = RenewBookForm(initial={'renewal_date':
proposed_renewal_date})

    context = {

        'form': form,

        'book_instance': book_instance,

    }

    return render(request,
'catalog/book_renew_librarian.html', context)
```

template ای که در view

(`/catalog/templates/catalog/book_renew_librarian.html`) به آن اشاره شده را

ایجاد کرده و کد زیر را در آن کپی کنید:

```
{% extends "base_generic.html" %}

{% block content %}

    <h1>Renew: {{ book_instance.book.title }}</h1>

    <p>Borrower: {{ book_instance.borrower }}</p>

    <p{% if book_instance.is_overdue %} class="text-danger"{%
endif %}>Due date: {{ book_instance.due_back }}</p>


    <form action="" method="post">

        {% csrf_token %}

        <table>

            {{ form.as_table }}

        </table>

        <input type="submit" value="Submit">

    </form>

{% endblock %}
```

آموزش Django

در آموزش های قبلی با اکثر اجزا این template آشنا شده اید. template اصلی را extend کرده و تعاریف بلوک content را تغییر میدهم. میتوانیم برای لیست عنوان کتاب ها، کتاب های امانت گرفته شده و تاریخ بازگشت اولیه از `{{ book_instance }}` (و متغیر هایش) نیز استفاده کنیم ، زیرا به شی context تابع render() پاس داده شده اند.

این form ، کد نسبتا ساده ای دارد. ابتدا، با مشخص کردن مکان submit ، فرم (action) و متدی که برای submit داده ها استفاده میشود (در مثال ما HTTP POST) و تگ های فرم را تعریف میکنیم.

اگر [HTML Form overview](#) موجود در بالای صفحه را دوباره فراخوانی کنید، یک action خالی به این معناست که داده ها به صفحه با URL حال حاضر ارسال میشوند (که عمکرد مورد نظر ماست !). داخل تگ ها ورودی submit را تعریف میکنیم تا هر کاربر بتواند با فشردن آن، داده ها را submit کند. `{% csrf_token %}` اضافه شده به داخل تگ های فرم بخشی از Django cross site forgery protection میباشد.

توجه

`{% csrf_token %}` را به تمامی template های Django که از POST برای submit داده ها استفاده میکنند اضافه کنید. با این کار احتمال hijack شدن فرم ها توسط کاربران مخرب، کاهش میابد.

آموزشگاه تحلیل داده

تنها متغیر template، `{{ form }}` باقی مانده است، که در context dictionary به template پاس داده شده است و به شکل زیر render پیش فرض، تمامی فیلد های فرم شامل برجسب ها، widget ها و متون کمکی آنها را، فراهم میکند:

```
<tr>

<th><label for="id_renewal_date">Renewal date:</label></th>

<td>
```



```
<input id="id_renewal_date" name="renewal_date"
type="text" value="2016-11-08" required>

<br>

<span class="helptext">Enter date between now and 4 weeks
(default 3 weeks).</span>

</td>

</tr>
```

توجه

به صورت پیش فرض ، هر فیلد در سطر جدول خودش تعریف میشود (که ممکن است ، با توجه به وجود تنها یک فیلد ، در مثال ما چندان مشخص نباشد). اگر از `reference` متغیر `template` ، `{{ form.as_table }}` نیز استفاده کنید، `render` به طریق مشابهی انجام میشود.

اگر تاریخ نامعتبری وارد کنید، (به شکل زیر) با لیستی از خطاهای `render` شده توسط صفحه مواجه میشوید:

```
<tr>

<th><label for="id_renewal_date">Renewal date:</label></th>

<td>

<ul class="errorlist">

<li>Invalid date - renewal in past</li>

</ul>
```

```
<input id="id_renewal_date" name="renewal_date"
type="text" value="2015-11-08" required>

<br>

<span class="helptext">Enter date between now and 4
weeks (default 3 weeks).</span>

</td>

</tr>
```

روش های دیگر استفاده از form template variable

با استفاده از `{{ form.as_table }}` به شکل بالا، هر فیلد به عنوان سطری از جدول `render` میشود. همچنین میتوانید هر فیلد را به عنوان یک `item` از لیست (با استفاده از `{{ form.as_ul }}`) و یا یک پاراگراف (با استفاده از `{{ form.as_p }}`) نیز `render` کنید.

همچنین میتوان، با `index` کردن `property` ها با کمک `dot notation`، `render` هر بخش از فرم را نیز به شکل کامل کنترل کرد. پس، برای مثال میتوانیم به شکل جداگانه به برخی از `item` های فیلد `renewal_date` دسترسی پیدا کنیم:

- `{{ form.renewal_date }}` : تمام فیلد
- `{{ form.renewal_date.errors }}` : لیست خطاها
- `{{ form.renewal_date.id_for_label }}` : `id` برچسب
- `{{ form.renewal_date.help_text }}` : فیلد متن کمکی

برای مشاهده مثال های بیشتر `render` دستی فرم ها در `template` و ایجاد حلقه دینامیک در فیلد های `template`، [Working with forms > Rendering fields manually](#) را مطالعه کنید.

تست صفحه

اگر چالش (احراز هویت کاربر و مجوز ها) را با موفقیت به اتمام رسانده باشید، لیستی از کتاب های امانت داده شده کتابخانه در اختیار دارید که تنها برای کارکنان کتابخانه قابل مشاهده است. میتوانیم با استفاده از کد زیر در کنار هر item این لیست ، لینکی به صفحه تجدید مهلت کتاب قرار دهیم :

```
{% if perms.catalog.can_mark_returned %}- <a href="{% url 'renew-book-librarian' bookinst.id %}">Renew</a> {% endif %}
```

توجه

توجه کنید که login آزمایشی شما باید دارای مجوز `catalog.can_mark_returned` ، برای دسترسی به صفحه تجدید کتاب، باشد (مثلا اکانت `superuser`).

همچنین میتوانید به شکل دستی نیز یک URL آزمایشی، مثلا به شکل <http://127.0.0.1:8000/catalog/book/<bookinstance id>/renew/> بسازید (برای یافتن یک `bookinstance_id` معتبر، میتوانید به صفحه مشخصات یک کتاب در کتابخانه رفته و فیلد `id` آن را کپی کنید).

شکل نهایی

اگر مراحل قبل را با موفقیت به پایان رسانده باشید، فرم پیش فرض به شکل زیر خواهد بود :

← → ↻ ⓘ 127.0.0.1:8000/catalog/book/d2ad0f63-82b0-46ac-8511-c89b76756a6b/renew/

[Home](#)
[All books](#)
[All authors](#)

Renew: The Wise Man's Fear
Borrower: superman
Due date: Oct. 12, 2016
Renewal date:
Enter date between now and 4 weeks (default 3 weeks).

[User: superman](#)
[My Borrowed](#)
[Logout](#)

[Staff](#)
[All borrowed](#)



با وارد کردن یک مقدار نامعتبر، فرم به شکل زیر تغییر میکند:

← → ↻ ⓘ 127.0.0.1:8000/catalog/book/d2ad0f63-82b0-46ac-8511-c89b76756a6b/renew/

← → ↻ ⓘ 127.0.0.1:8000/catalog/book/d2ad0f63-82b0-46ac-8511-c89b76756a6b/renew/

[Home](#)
[All books](#)
[All authors](#)

Renew: The Wise Man's Fear
Borrower: superman
Due date: Oct. 12, 2016

- Invalid date - renewal in past

Renewal date:
Enter date between now and 4 weeks (default 3 weeks).

[User: superman](#)
[My Borrowed](#)
[Logout](#)

[Staff](#)
[All borrowed](#)



و لینک تجدید مهلت کتاب در لیست کتاب های امانت داده شده نیز به شکل زیر خواهد بود:

[Home](#)
[All books](#)
[All authors](#)

User: superman
[My Borrowed](#)
[Logout](#)

All Borrowed Books

- [The Wise Man's Fear \(Oct. 12, 2016\)](#) - [superman](#) - [Renew](#)
- [The Name of the Wind \(Nov. 8, 2016\)](#) - [superman](#) - [Renew](#)
- [The Dueling Machine \(Nov. 8, 2016\)](#) - [johnc](#) - [Renew](#)

[Staff](#)
[All borrowed](#)

ModelForms

روش بالا، روش بسیار انعطاف پذیری برای ساخت یک کلاس **Form** است زیرا امکان ایجاد هرگونه فرمی با مدل های دلخواه را برای ما فراهم میکند.

اگر تنها نیاز به نگاشت فیلد های یک مدل در یک فرم دارید، مدل اکثر اطلاعات مورد نیاز شما شامل فیلد ها، برچسب ها، متن های کمکی و... را تعریف میکند. به جای نوشتن تعاریف مدل در فرم میتوانید به سادگی از کلاس کمکی [ModelForm](#) استفاده کرده و فرم را با استفاده از مدل خود بسازید. **ModelForm** ایجاد شده، مانند تمامی **Form** های دیگر، در **view** ها قابل استفاده است.

یک **ModelForm** ساده، محتوی فیلد های یکسانی با **RenewBookForm** مان، در بخش زیر نمایش داده شده است. برای ایجاد فرم تنها لازم است **Class Meta** را به همراه **Model** مرتبط (**BookInstance**) و لیستی از فیلد های مدل را اضافه کنید تا درون فرم قرار بگیرند.

```
from django.forms import ModelForm
```

```
from catalog.models import BookInstance
```

```
class RenewBookModelForm(ModelForm):  
  
    class Meta:  
  
        model = BookInstance  
  
        fields = ['due_back']
```

توجه

میتوانید با استفاده از `fields = '__all__'` تمامی فیلدها را قرار داده و یا به کمک `exclude` به جای `fields` فیلدهای حذفی را مشخص کنید.

البته استفاده از روش های بالا توصیه نمیشود زیرا با استفاده از این روش ها فیلدهای جدید اضافه شده به مدل، بدون انجام اقدامات لازم امنیتی توسط `developer`، به فرم اضافه میشوند.

توجه

این روش در مثال ما (به علت وجود تنها یک فیلد) چندان ساده تر از روش قبلی به نظر نمیرسد، اما اگر تعداد فیلدهای استفاده شده زیاد باشند، استفاده از این روش حجم کد را مقدار قابل توجهی می کاهد.

بقیه اطلاعات از تعاریف فیلدهای مدل (مانند برچسب ها، `widget`ها، متن های کمکی، خطا ها) گرفته میشود. در صورت لزوم تغییر فیلدها میتوانید با `override` کردن `class Meta`، ایجاد لیستی از فیلدهایی که تمایل به تغییر آنها دارید و مقادیر جدید آنها، تغییرات لازم را انجام دهید. برای مثال، در این فرم میتوانیم (به جای استفاده از برچسب پیش فرض : Due Back) از "Renewal Date" استفاده کرده و متن کمکی را نیز به متنی مناسب و مشخص تغییر دهیم. `Meta` زیر نحوه `override` کردن این فیلدها را نمایش میدهد. با روشی مشابه میتوانید `Widget` ها و پیام های خطا را نیز تغییر دهید.

```
class Meta:

    model = BookInstance

    fields = ['due_back']

    labels = {'due_back': _('New renewal date')}

    help_texts = {'due_back': _('Enter a date between now and
4 weeks (default 3).')}
```

validation را نیز میتوانید(با رویکرد مشابه برای فرم های معمول، تعریف یک تابع به نام clean_field_name() و ایجاد یک ValidationError exception برای مقادیر نامعتبر) به فرم خود اضافه کنید.

تنها تفاوت این فرم با فرم اولیه ای که ایجاد کردیم تغییر نام فیلد مدل به due_back است (به جای "renewal_date").

دقت کنید که با توجه به اینکه فیلد متناظر در BookInstance، due_back نام دارد، این تغییر ضروری است.

```
from django.forms import ModelForm

from catalog.models import BookInstance

class RenewBookModelForm(ModelForm):

    def clean_due_back(self):

        data = self.cleaned_data['due_back']
```

```
# Check if a date is not in the past.

if data < datetime.date.today():

    raise ValidationError(_('Invalid date - renewal in
past'))

# Check if a date is in the allowed range (+4 weeks
from today).

if data > datetime.date.today() +
datetime.timedelta(weeks=4):

    raise ValidationError(_('Invalid date - renewal
more than 4 weeks ahead'))

# Remember to always return the cleaned data.

return data

class Meta:

    model = BookInstance

    fields = ['due_back']

    labels = {'due_back': _('Renewal date')}
```



```
help_texts = {'due_back': _('Enter a date between now  
and 4 weeks (default 3).')}
```

کلاس `RenewalBookModelForm` بالا عملکردی معادل با فرم اولیه مان ، `RenewalBookForm` ، دارد.

میتوانید در هر مکانی که در حال حاضر از `RenewalForm` استفاده میکنید، `RenewalBookModelForm` را `import` کرده و جایگزین کنید.

اما دقت کنید که برای این کار باید (مانند تعریف فرم دوم:

```
RenewBookModelForm(initial={'due_back':  
proposed_renewal_date})
```

نام متغیر متناظر در فرم را از `renewal_date` به `due_back` تغییر دهید.

Generic editing views

الگوریتم `form handling` مورد استفاده ما در مثال `function view` بالا، یک الگو بسیار متداول در `form handling view` ها میباشد.

Django با ایجاد [generic editing views](#) به منظور ساخت، حذف و ویرایش `view` های مبنی بر مدل، بخش زیادی از این کارهای تکراری را برای ما خلاصه میکند. `Generic editing view` ها رفتار `view` را مدیریت کرده و به شکل خودکار و با استفاده از مدل، `form class` (`ModelForm`) های شما را نیز ایجاد میکنند.

توجه

علاوه بر `editing view` ها که در این بخش بررسی شد، کلاس [FormView](#) نیز موجود است که در مقایسه "انعطاف پذیری" و "سختی کد نویسی" جایی بین `function view` و `generic view` های دیگر قرار میگیرد. در زمان استفاده از `FormView` ها، همچنان لازم است که `Form` هایتان را خودتان

آموزش Django

ایجاد کنید اما نیازی نیست تمام الگوهای **form handling** را پیاده سازی کنید. به جای این کار تنها باید یک تابع، که پس از تایید ورودی های **submit** شده فراخوانی میشود، بسازید.

در این بخش با استفاده از **generic editing view** ها و پیاده سازی دوباره بخش هایی از **Admin site**، صفحاتی با قابلیت های ایجاد، ویرایش و حذف نویسندگان از کتابخانه استفاده میکنیم (با این روش میتوان قابلیت های **Admin** را با روشی انعطاف پذیر تر از **admin site** ارائه کرد).

View ها

فایل **View** را باز کرده (**locallibrary/catalog/views.py**) و قطعه کد زیر را به انتهای آن اضافه کنید:

```
from django.views.generic.edit import CreateView, UpdateView, DeleteView

from django.urls import reverse_lazy

from catalog.models import Author

class AuthorCreate(CreateView):

    model = Author

    fields = ['first_name', 'last_name', 'date_of_birth', 'date_of_death']

    initial = {'date_of_death': '11/06/2020'}
```

```
class AuthorUpdate(UpdateView):

    model = Author

    fields = '__all__' # Not recommended (potential security
                       # issue if more fields added)

class AuthorDelete(DeleteView):

    model = Author

    success_url = reverse_lazy('authors')
```

همانطور که مشاهده میکنید، برای ایجاد، به روز رسانی و یا حذف **view** ها باید به ترتیب از **CreateView**، **UpdateView** و **DeleteView** مشتق گیری (derive) کرده و سپس مدل های متناظر را تعریف کنید.

برای به "update" و "create" باید فیلدهایی که در فرم نمایش داده میشوند را نیز مشخص کنید (با استفاده از سینکسی مشابه **ModelForm**). در این بخش، روش لیست کردن آنها به شکل جداگانه و همچنین سینتکس ایجاد لیستی از "همه" فیلدها را به شما نشان میدهم. میتوانید، با استفاده از یک **dictionary** از مقادیر جفت های **field_name** (در این بخش ما به هدف نمایش از تاریخ فوت استفاده کرده ایم که میتوانید آن را حذف کنید!)، مقادیر اولیه ای نیز برای هر فیلد ایجاد کنید. به شکل پیش فرض این صفحات ، در صورت موفقیت در عملیات، به صفحه ی جدیدی که مدل تازه تغییر یافته/ایجاد شده را نمایش میدهد، هدایت میشوند. میتوانید با تعریف پارامتر **success_url** (به روش مشابه کلاس **AuthorDelete**) صفحه را به مکان جدیدی هدایت کنید.

Django آموزش

کلاس AuthorDelete نیازی به نمایش هیچ فیلدی ندارد اما، همچنان باید success_url را تعریف کنید، زیرا مقدار پیش فرضی برای استفاده Django وجود ندارد. در این بخش ما با استفاده از تابع [reverse_lazy\(\)](#)، پس از حذف یک نویسنده، به صفحه لیست نویسندگان تغییر مسیر میدهیم. تابع `reverse_lazy()` نسخه تنبلی (!) از تابع `reverse()` است که به علت استفاده از یک URL به یک `class based view attribute` از آن استفاده میکنیم.

Template ها

view های create و update به شکل پیش فرض از `template` هایی استفاده میکنند که به شکل `model_name_form.html` نام گذاری میشوند (میتوانید با استفاده از فیلد `template_name_suffix` در `view`، پسوند نام `view` را تغییر داده و پسوند دیگری به جای `_form`، مثلاً `template_name_suffix = '_other_suffix'`، استفاده کنید).

فایل `template` زیر را ایجاد کرده و متن زیر را در آن کپی کنید
: `locallibrary/catalog/templates/catalog/author_form.html`

```
{% extends "base_generic.html" %}

{% block content %}

    <form action="" method="post">

        {% csrf_token %}

        <table>

            {{ form.as_table }}

        </table>
```

```
<input type="submit" value="Submit">

</form>

{% endblock %}
```

این بخش عملکردی مشابه فرم های پیشین داشته و فیلدها را با استفاده از یک جدول `render` میکند. توجه کنید که برای اطمینان از مقاومت فرم در برابر حملات `CSRF`، دوباره `{% csrf_token %}` را تعریف میکنیم.

`view` حذف `(delete)`، انتظار یافتن `template` ای با فرمت نام `model_name_confirm_delete.html` (که همچنان میتوانید با استفاده از `template_name_suffix` در `view`، پسوند آن را تغییر دهید) دارد. این فایل `template` را ایجاد کرده و متن زیر را در آن کپی کنید : `locallibrary/catalog/templates/catalog/author_confirm_delete.html`

```
{% extends "base_generic.html" %}

{% block content %}

<h1>Delete Author</h1>

<p>Are you sure you want to delete the author: {{ author }}?</p>

<form action="" method="POST">
```

```
{% csrf_token %}

<input type="submit" value="Yes, delete.">

</form>

{% endblock %}
```

تنظیمات URL

فایل URL configuration خود را باز کرده (`locallibrary/catalog/urls.py`) و تنظیمات زیر را در انتهای آن قرار دهید :

```
urlpatterns += [

    path('author/create/', views.AuthorCreate.as_view(),
name='author-create'),

    path('author/<int:pk>/update/',
views.AuthorUpdate.as_view(), name='author-update'),

    path('author/<int:pk>/delete/',
views.AuthorDelete.as_view(), name='author-delete'),

]
```

در این بخش مطلب چندان تازه ای وجود ندارد! مشاهده میکنید که **view** ها کلاس هستند و باید با استفاده از `as_view()` فراخوانی شوند. همچنین باید بتوانید در هر بخش الگو های URL را تشخیص دهید. برای مقدار کلید اصلی دریافت شده از نام **pk** استفاده میکنیم زیرا کلاس **view** پارامتری با این نام دریافت میکند.

حالا صفحات ایجاد، به روز رسانی و حذف نویسندگان مان آماده تست هستند (اینجا از مرحله اضافه کردن آنها به sidebar چشم پوشی میکنیم، اما میتوانید آنها را به sidebar نیز اضافه کنید).

توجه

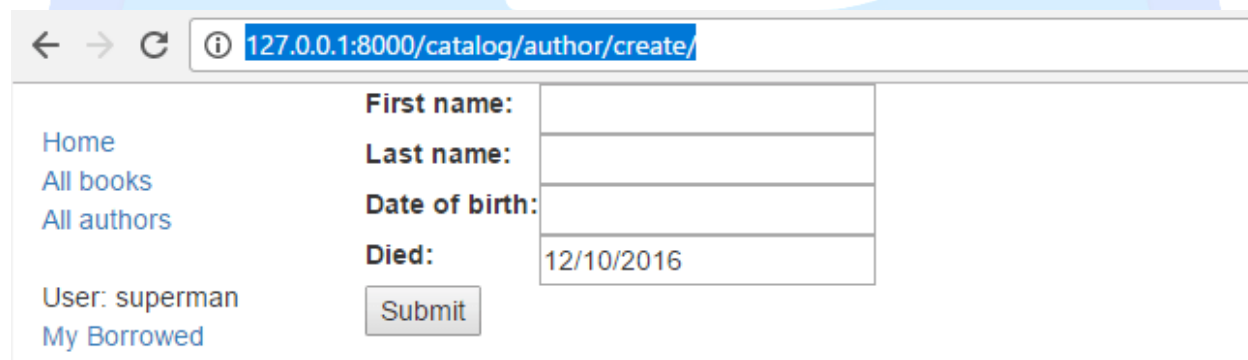
اگر خواننده دقیقی باشید، حتما دقت کرده اید که ما دسترسی کاربران تایید هویت نشده را به این صفحات محدود نکرده ایم! این بخش از کار را به عنوان تمرین به شما واگذار میکنیم (راهنمایی : میتوانید با کمک `PermissionRequiredMixin` یک مجوز جدید ایجاد کرده و یا از مجوز `can_mark_returned` استفاده کنید).

تست صفحه

ابتدا با حسابی که مجوز های لازم شما برای دسترسی به صفحات ویرایش نویسندگان را دارد، به سایت وارد شوید.

به صفحه ویرایش نویسندگان بروید :

<https://127.0.0.1:8000/catalog/author/create> که باید ظاهری مشابه تصویر زیر داشته باشد.

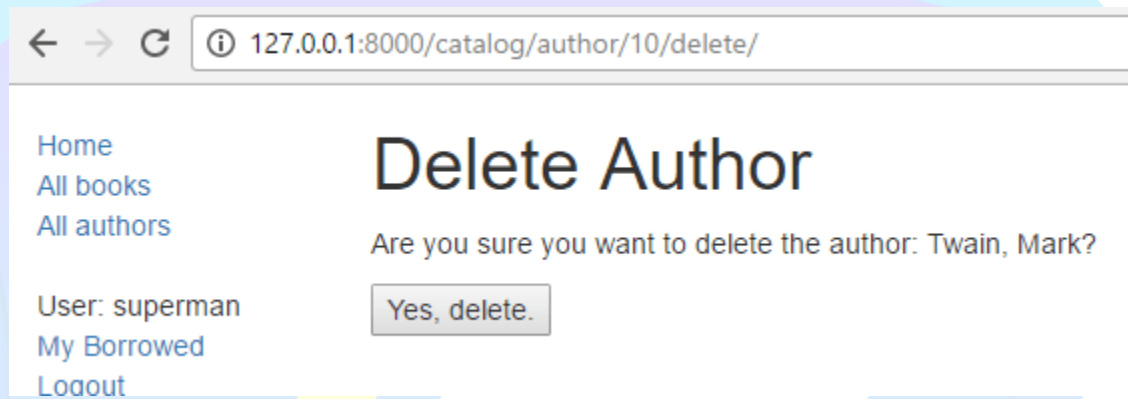


مقادیر فیلد ها را وارد کرده و **Submit** را انتخاب کنید تا رکورد نویسنده ذخیره شود. حالا باید قادر باشید صفحه مشخصات نویسنده جدید را ، در URL ای شبیه به <http://127.0.0.1:8000/catalog/author/10> مشاهده کنید.

Django آموزش

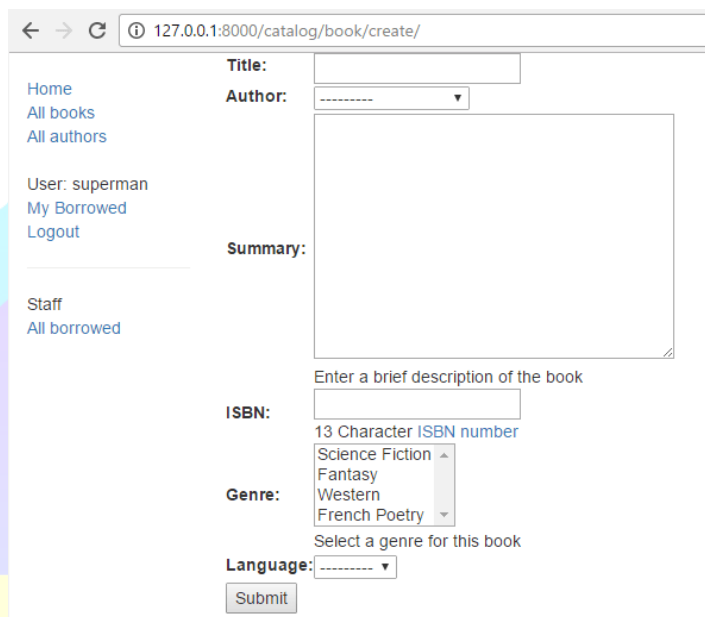
میتوانید با ضمیمه کردن `/update/` به انتهای URL صفحه `detail view` مانند <http://127.0.0.1:8000/catalog/author/10/update/> ویرایش رکورد ها را نیز تست کنید که به علت تشابه آن با صفحه `create`، تصویر آن را نمایش نمیدهیم!

در نهایت، میتوانید با ضمیمه کردن `delete` به انتهای URL صفحه `detail view` مانند <http://127.0.0.1:8000/catalog/author/10/delete/> آن صفحه را حذف کنید. Django باید صفحه زیر را به شما نمایش دهد. "Yes, delete." را انتخاب کنید تا رکورد از لیست نویسندگان حذف شود.



خود را به چالش بکشید

فرم هایی برای حذف، ایجاد و ویرایش رکورد های `Book` ایجاد کنید. میتوانید از ساختاری دقیقا مانند ساختار استفاده شده برای `Authors` استفاده کنید. اگر `template` ، `book_form.html` تان دقیقا نسخه مشابه با `author_form.html` است و فقط نام متفاوتی دارد، صفحه `create` کتابتان به شکل زیر خواهد بود :



← → ↻ 127.0.0.1:8000/catalog/book/create/

Home
All books
All authors

User: superman
My Borrowed
Logout

Staff
All borrowed

Title:

Author:

Summary:

Enter a brief description of the book

ISBN:

13 Character ISBN number

Genre:

Science Fiction
Fantasy
Western
French Poetry

Select a genre for this book

Language:

Submit

خلاصه

ساخت و مدیریت فرم ها میتواند فرآیند بسیار پیچیده ای باشد! Django با فراهم کردن مکانیزم های گرامری برای تعریف، render و تایید فرم ها، استفاده از آنها را برایمان بسیار ساده میکند. علاوه بر این generic form editing view های موجود در Django تقریبا تمام عملیات لازم برای تعریف صفحه، ویرایش و حذف رکورد های متناظر با یک model instance را برای ما انجام میدهند.

فرم ها قابلیت های بسیار زیادی دارند (که میتوانید در [see also](#) آنها را مطالعه کنید) و حالا شما قادر هستید یک فرم ساده را به وب سایت خود اضافه کرده و آن را مدیریت کنید.