

طراحی سایت با جنگو

(وبلاگ- وبسایت شبکه اجتماعی

وب سایت فروشگاهی و

(CMS



Django Tutorial

* Website-Blog- CMS

مترجم: علی عوض زاده

. ساختن یک برنامه وبلاگ

نصب جنگو

ایجاد یک محیط جدا شده پایتون

نصب جنگو با پیپ

ایجاد اولین پروژه خود

اجرای سرور توسعه

تنظیمات پروژه

پروژه ها و برنامه های کاربردی

ایجاد یک برنامه

طراحی شمای داده های وبلاگ

برنامه خود را فعال کنید

ایجاد و اعمال مهاجرت

ایجاد یک سایت مدیریت برای مدل های خود

ایجاد یک ابرقدرت

سایت مدیریت جنگو

مدل های خود را به سایت مدیریت اضافه کنید

سفارشی کردن نحوه نمایش مدلها

همکاری با QuerySet و مدیران

ایجاد اشیاء

به روزرسانی اشیاء

بازیابی اشیاء

استفاده از filter()

استفاده از remove

استفاده از order_by()

حذف اشیاء

وقتی QuerySets ارزیابی می شود

ایجاد مدیران مدل

لیست ساختمان و نمایش جزئیات

ایجاد لیست و نمایش جزئیات

اضافه کردن الگوهای URL برای بازدیدهای شما

URL های متعارف برای مدل ها

ایجاد الگوهای برای دیدگاه های شما

اضافه کردن صفحه بندی

استفاده از نماهای کلاس محور

خلاصه

2. تقویت وبلاگ خود با ویژگی های پیشرفته

اشتراک گذاری پست ها از طریق ایمیل

ایجاد فرم با جنگو

رسیدگی به اشکال در نماها

ارسال ایمیل با Django

ارائه فرم در قالب

ایجاد سیستم کامنت

ایجاد فرم از مدل ها

رسیدگی به ModelForms در نماها

افزودن نظرات به الگوی جزئیات ارسال

اضافه کردن قابلیت برچسب زدن

بازیابی پست ها با تشابه

خلاصه

3. گسترش برنامه وبلاگ خود

ایجاد برچسب ها و فیلترهای قالب های سفارشی

ایجاد برچسب های قالب سفارشی

ایجاد فیلترهای قالب سفارشی

نقشه سایت را به سایت خود اضافه کنید

ایجاد فید برای پست های وبلاگ خود

جستجوی متن کامل را به وبلاگ خود اضافه کنید

نصب PostgreSQL

جستجوی ساده جستجو

در حال جستجو در مقابل چندین زمینه

ایجاد نمای جستجو

نتایج تقویت شده و رتبه بندی

پرس و جوهای وزنه برداری

جستجو با شباهت trigram

سایر موتورهای جستجو متن کامل

خلاصه

4- ساختن یک وب سایت اجتماعی

ایجاد یک پروژه وب سایت اجتماعی

شروع پروژه وب سایت اجتماعی خود

استفاده از چارچوب احراز هویت Django

ایجاد نمای ورود

استفاده از نماهای تأیید هویت Django

نمایش ورود و خروج

تغییر نمایش گذرواژه

بازنشانی نمایش گذرواژه

ثبت نام کاربر و پروفایل کاربر

ثبت نام کاربر

گسترش مدل کاربر

استفاده از یک مدل کاربر سفارشی

استفاده از چارچوب پیام ها

ایجاد یک پس زمینه تأیید هویت سفارشی

اضافه کردن احراز هویت اجتماعی به سایت شما

احراز هویت با استفاده از Facebook

تأیید اعتبار با استفاده از توییتر

تأیید اعتبار با استفاده از Google

خلاصه

5- به اشتراک گذاری محتوا در وب سایت شما

ایجاد یک وب سایت نشانک تصویر

ساختن مدل تصویر

ایجاد روابط بسیار زیاد

ثبت مدل تصویر در سایت مدیریت

ارسال مطالب از وب سایت های دیگر

تمیز کردن زمینه های فرم

نادیده گرفتن روش save() در ModelForm

ساخت یک نشانک با jQuery

ایجاد نمای جزئی برای تصاویر

ایجاد ریز عکسها با استفاده از تصاویر کوچک

افزودن اقدامات AJAX با jQuery

بارگیری jQuery

درخواست متقابل درخواست جعلی در درخواست های AJAX

انجام درخواستهای AJAX با jQuery

ایجاد دکوراتورهای سفارشی برای نماهای شما

افزودن صفحه بندی AJAX به نماهای لیست شما

خلاصه

6. پیگیری عملکردهای کاربر

ساختن یک سیستم پیروی

ایجاد روابط بسیار زیاد با یک واسطه

مدل

ایجاد لیست و نمایش جزئیات برای پروفایل کاربر

ایجاد یک نمای AJAX برای دنبال کردن کاربران

ایجاد یک برنامه جریان فعالیت عمومی

استفاده از چارچوب محتوا

روابط عمومی را به مدل‌های خود اضافه کنید

اجتناب از اقدامات تکراری در جریان فعالیت

افزودن اقدامات کاربر به جریان فعالیت

نمایش جریان فعالیت

بهینه سازی QuerySets که شامل اشیاء مرتبط می‌باشد

استفاده از `Select_related()`

استفاده از `prefetch_related()`

ایجاد الگوهای عمل

استفاده از سیگنال‌ها برای شمارش آزار دهنده تعداد

کار با سیگنال ها

کلاس پیکربندی برنامه

استفاده از Redis برای ذخیره نماهای مورد

نصب Redis

استفاده از Redis با پایتون

ذخیره نماهای مورد در Redis

ذخیره رتبه بندی در Redis

مراحل بعدی با Redis

خلاصه

7. ساخت فروشگاه آنلайн

ایجاد یک پروژه فروشگاه آنلайн

ایجاد مدل‌های فروشگاه محصولات

ثبت مدل‌های کاتالوگ در سایت سرپرست

نمایش کاتالوگ ساختمان

ایجاد الگوهای کاتالوگ

ساخت سبد خرید

استفاده از جلسات جنگو

تنظیمات جلسه

منقضی شدن جلسه

ذخیره سبد خرید در جلسات

ایجاد نماهای سبد خرید

افزودن موارد به سبد خرید

ساختن یک الگوی برای نشان دادن سبد خرید

افزودن محصولات به سبد خرید

به روزرسانی مقادیر محصول در سبد خرید

ایجاد یک پردازنده زمینه برای سبد خرید فعلی

پردازنده های متن

سبد خرید را در متن درخواست تنظیم کنید

ثبت سفارشات مشتری

ایجاد مدل های سفارش

از جمله مدل های سفارش در سایت مدیریت

ایجاد سفارشات مشتری

انجام کارهای ناهمزمان با celery

نصب celery

نصب RabbitMQ

celery را به پروژه خود اضافه کنید

وظایف ناهمزمان را به برنامه خود اضافه کنید

نظرات بر celery

خلاصه

8- مدیریت پرداختها و سفارشات

یکپارچگی دروازه پرداخت

ایجاد یک حساب کاربری Braintree sandbox

نصب ماژول Braintree Python

یکپارچه کردن دروازه پرداخت

ادغام Braintree با استفاده از زمینه های میزبانی شده

تست پرداختها

زندگی می کنند

صادرات سفارشات به پرونده های CSV

افزودن اقدامات سفارشی به سایت مدیریت

گسترش سایت سرپرست با نماهای سفارشی

تولید فاکتورهای PDF به صورت پویا

نصب WeasyPrint

ایجاد یک قالب PDF

ارائه فایل‌های PDF

ارسال فایل‌های PDF از طریق ایمیل

خلاصه

9. گسترش فروشگاه شما

ایجاد سیستم کوپن

ساخت مدل‌های کوپن

اعمال کوپن به سبد خرید

اعمال کوپن به سفارشات

افزودن بین‌المللی و بومی سازی

بین‌المللی شدن با جنگو

تنظیمات بین‌المللی سازی و بومی سازی

دستورات مدیریت بین‌المللی

چگونه ترجمه‌ها را به یک پروژه Django اضافه کنیم

چگونه جنگو زبان فعلی را تعیین می‌کند

آماده سازی پروژه ما برای بین‌المللی شدن

ترجمه کد پایتون

ترجمه های استاندارد

ترجمه های تنبل

ترجمه از جمله متغیرها

اشکال جمع در ترجمه

ترجمه کد خود

ترجمه الگوهای

برچسب قالب {٪trans٪}

برچسب قالب {٪blocktrans٪}

ترجمه قالب فروشگاه

استفاده از رابط ترجمه Rosetta

ترجمه های فازی

الگوهای URL برای بین المللی

افزودن یک پیشوند زیان به الگوهای URL

ترجمه الگوهای URL

اجازه دادن به کاربران برای تغییر زیان

ترجمه مدل ها با django-parler

نصب django-parler

زمینه های مدل ترجمه

ادغام ترجمه ها در دولت

سایت

ایجاد مهاجرت برای ترجمه مدل

تطبیق نمایش برای ترجمه

بومی سازی قالب

استفاده از django-localflavor برای تأیید صحت زمینه های فرم

ساختن موتور توصیه ای

توصیه محصولات بر اساس خریدهای قبلی

خلاصه

10. ایجاد یک بستر الکترونیکی یادگیری

تنظیم پروژه یادگیری الکترونیکی

ساخت مدل های دوره

ثبت مدل ها در سایت اداره

استفاده از لوازم جانبی برای تهیه &؛ داده های اولیه مدل

ایجاد مدل هایی برای مطالب متنوع

استفاده از ارث مدل

مدلهای انتزاعی

وراثت مدل چند جدول

مدل های پروکسی

ایجاد مدل های محتوا

ایجاد زمینه های مدل های سفارشی

افزودن سفارش به اشیاء مژول و محتوا

ایجاد CMS

اضافه کردن یک سیستم تأیید اعتبار

ایجاد الگوهای تأیید اعتبار

ایجاد نماهای کلاس محور

استفاده از مخلوط ها برای نمایش کلاس مبتنی بر کار با گروه ها و مجوزها

محدود کردن دسترسی به نماهای مبتنی بر کلاس

مدیریت مژول ها و محتوا

استفاده از مجموعه های فرم برای مژول های دوره

افزودن محتوا به مژول های دوره

مدیریت مژول ها و محتویات

تنظیم مجدد مازول ها و مطالب

استفاده از mixins django-braces

خلاصه

11. ارائه و ذخیره محتوای

نمایش دوره ها

اضافه کردن ثبت نام دانشجویی

ایجاد نمای ثبت نام دانشجویی

ثبت نام در دوره ها

دسترسی به مطالب دوره

ارائه انواع مختلف محتوا

استفاده از چارچوب کش

پس زمینه های حافظه نهان موجود

نصب Memcached

تنظیمات حافظه نهان

افزودن Memcached به پروژه شما

نظرارت بر Memcached

سطح حافظه نهان

استفاده از API حافظه نهان سطح پایین

ذخیره سازی بر اساس داده های پویا

قطعات قالب ذخیره

ذخیره نماها

استفاده از حافظه نهانگاه در هر سایت

خلاصه

12. ساخت API

ساختن یک API RESTful

چارچوب نصب جنگو REST

تعریف سریال ها

درک پارس و رندرها

لیست ساختمان و نمایش جزئیات

ایجاد سریالایزرهای تو در تو

ایجاد نماهای سفارشی

رسیدگی به تأیید اعتبار

افزودن مجوز به بازدیدها

ایجاد مجموعه نمایش و روتور

افزودن اقدامات اضافی برای مشاهده مجموعه ها

ایجاد مجوزهای سفارشی

سریال کردن مطالب دوره

خلاصه

13. زنده بودن

ایجاد یک محیط تولید

مدیریت تنظیمات برای چندین محیط

استفاده از PostgreSQL

بررسی پروژه خود

خدمت جنگو از طریق WSGI

نصب uWSGI

پیکربندی uWSGI

نصب NGINX

محیط تولید

پیکربندی NGINX

ارائه دارایی های استاتیک و رسانه ای

امن سازی اتصالات با SSL

ایجاد گواهی SSL

پیکربندی NGINX برای استفاده از SSL

پیکربندی پروژه ما برای SSL

ایجاد یک واسطه سفارشی

ایجاد واسطه زیر دامنه

در خدمت چندین زیر دامنه با NGINX

اجرای دستورات مدیریت سفارشی

خلاصه

آنچه این کتاب را در برمی گیرد

فصل 1 ، ایجاد یک برنامه وبلاگ ، شما را با ایجاد یک برنامه وبلاگ به چهارچوب معرفی می کند. شما برای نمایش پست های وبلاگ ، مدل های اولیه وبلاگ ، نماها ، قالب ها و URL ها ایجاد خواهید کرد. شما یاد می گیرید که چگونه Django ORM QuerySets را با Django مسازید ، و سایت مدیریت Django را پیکربندی کنید.

فصل 2 ، تقویت وبلاگ شما با ویژگی های پیشرفته ، به شما می آموزد که چگونه فرمها و فرم های مدل را مدیریت کنید ، ایمیل بفرستید با Django و ادغام برنامه های شخص ثالث. شما یک سیستم نظر برای پست های وبلاگ خود پیاده سازی می کنید و به کاربران خود اجازه می دهید پست ها را از طریق ایمیل به اشتراک بگذارند. این فصل همچنین شما را با روند ایجاد سیستم برچسب زدن راهنمایی می کند.

فصل 3 ، گسترش برنامه وبلاگ شما ، به بررسی چگونگی ایجاد برچسب ها و فیلترهای قالب های سفارشی می پردازد. این فصل همچنین به شما نشان می دهد که چگونه از چارچوب نقشه سایت و ایجاد فید RSS برای پست های خود استفاده کنید. با ساخت موتور جستجوگر با قابلیت جستجوی متن کامل PostgreSQL ، برنامه وبلاگ خود را تکمیل خواهید کرد.

فصل چهارم ، ایجاد یک وب سایت اجتماعی ، نحوه ساخت یک وب سایت اجتماعی را توضیح می دهد. برای ایجاد نماهای حساب کاربر از چارچوب تأیید اعتبار Django استفاده خواهید کرد. شما یاد می گیرید که چگونه با استفاده از شبکه های اجتماعی عمدی ، یک مدل پروفایل شخصی کاربر ایجاد کنید و احراز هویت اجتماعی را در پروژه خود ایجاد کنید.

فصل 5 ، به اشتراک گذاری محتوا در وب سایت شما ، به شما می آموزد که چگونه برنامه اجتماعی خود را به یک وب سایت نشانه گذاری تصویر تبدیل کنید. شما روابط بسیاری از بسیاری را برای مدل ها تعریف خواهید کرد و یک نشانک JavaScript AJAX را در ایجاد کرده و آن را در پروژه خود ادغام خواهید کرد. در این فصل نحوه تولید ریز عکسها و ایجاد دکوراتورهای سفارشی برای نمایش شما نشان داده می شود.

فصل 6 ، ردیابی عملکردهای کاربر ، نحوه ساختن یک سیستم دنبال کننده را برای کاربران نشان می دهد. با ایجاد یک برنامه کاربردی جریان کاربر ، وب سایت نشانک تصویر خود را کامل می کنید. شما یاد می گیرید که چگونه QuerySets را بهینه کنید ، و با سیگنال کارخواهید کرد. شما برای شمارش نماهای تصویر ، Redis را در پروژه خود ادغام خواهید کرد.

فصل 7 ، ساختن یک فروشگاه اینترنتی ، نحوه ایجاد یک فروشگاه آنلاین را بررسی می کند. شما مدل های کاتالوگ ایجاد خواهید کرد و با استفاده از جلسات جنگو سبد خرید ایجاد خواهید کرد. شما یک سبد خرید برای سبد خرید ایجاد خواهید کرد و یاد می گیرید که چگونه می توانید ارسال اعلان های ناهمزمان به کاربرانی که از Celery استفاده می کنند را پیاده سازی کنید.

در فصل هشتم ، مدیریت پرداخت ها و سفارشات ، نحوه ادغام دروازه پرداخت در فروشگاه شما توضیح داده شده است. همچنین سایت مدیریت را برای صادرات سفارشات به پرونده های CSV سفارشی خواهید کرد و فاکتورهای PDF را بصورت پویا تولید می کنید.

فصل 9 ، گسترش فروشگاه شما ، به شما می آموزد که چگونه یک سیستم کوپن برای اعمال تخفیف در سفارشات ایجاد کنید. این فصل چگونگی افزودن بین المللی به پروژه خود و نحوه ترجمه مدل ها را نشان می دهد. شما همچنین می توانید با استفاده از Redis یک موتور توصیه کننده محصول بسازید.

فصل 10 ، ایجاد یک بستر آموزش الکترونیکی ، شما را در ایجاد یک بستر یادگیری الکترونیکی راهنمایی می کند. شما وسایل خود را به پروژه خود اضافه می کنید ، از وراثت مدل استفاده می کنید ، زمینه های مدلها را دلخواه را ایجاد می کنید ، از نماهای مبتنی بر کلاس استفاده می کنید و گروه ها و مجوزها را مدیریت می کنید. شما یک سیستم مدیریت محتوا ایجاد خواهید کرد و فرمتهای را مدیریت خواهید کرد.

فصل 11 ، ارائه و ذخیره کردن مطالب ، نحوه ایجاد سیستم ثبت نام دانشجویی و مدیریت ثبت نام دانش آموزان در دوره ها را به شما نشان می دهد. شما مطالب متنوعی را ارائه می دهید و نحوه استفاده از چارچوب کش را یاد خواهید گرفت.

فصل 12 ، ساخت یک API ، ساخت یک API RESTful را برای پروژه خود با استفاده از چارچوب Django REST برسی می کند.

فصل سیزدهم ، Going Live ، نحوه تنظیم محیط تولید را با استفاده از uWSGI و NGINX و چگونگی تأمین امنیت آن با SSL نشان می دهد. در این فصل نحوه ساختن یک سرویس میان افزار سفارشی و ایجاد دستورات مدیریت سفارشی توضیح داده شده است.

پیشگفتار

یک چارچوب وب قدرتمند Django است که توسعه سریع و طراحی پاک و عملی را ترغیب می کند و یک منحنی یادگیری نسبتاً کم عمق را ارائه می دهد. این باعث می شود برای برنامه نویسان تازه کار جذاب باشد.

این کتاب شما را در طی کل مراحل توسعه برنامه های وب حرفه ای با جنگو راهنمایی می کند. این کتاب نه تنها مناسب ترین جنبه های چارچوب را در بر می گیرد ، بلکه به شما می آموزد که چگونه سایر فن آوری های محبوب را در پروژه های جنگو خود ادغام کنید. این کتاب شما را در ایجاد برنامه های کاربردی در دنیای واقعی ، حل مشکلات مشترک و اجرای بهترین شیوه ها با رویکرد گام به گام دنبال می کند که به راحتی امکان پذیر است. پس از خواندن این کتاب ، شما درک خوبی از نحوه کار Django و نحوه ساخت برنامه های کاربردی وب پیشرفتی و کاربردی خواهید داشت.

این کتاب برای چه کسی است

این کتاب برای توسعه دهنده‌گان با دانش پایتون در نظر گرفته شده است که مایل به یادگیری جنگو به روشنی عملی هستند. شاید شما کاملاً تازه وارد جنگو باشید ، یا از قبل کمی هم می دانید اما می خواهید از آن بیشترین بهره را ببرید. این کتاب به شما کمک می کند تا با ساختن پروژه های عملی از ابتدا ، در زمینه های مرتبط با این چارچوب استاد باشید. برای خواندن این کتاب باید دانش قبلی را با مفاهیم برنامه نویسی آشنا کنید ، فرض بر این است که دانش قبلی از HTML و JavaScript فرض شده است.

برای به دست آوردن بیشترین استفاده از این کتاب توصیه می شود که دانش خوبی در رابطه با Python داشته باشید همچنین باید از HTML و JavaScript. قبل از خواندن این کتاب ، توصیه می شود قسمت های 1 تا 3 از آموزش رسمی اسناد جنگو را در <https://docs.djangoproject.com/fa/2.0/intro/tutorial01>.

کنوانسیون های مورد استفاده

تعدادی کنوانسیون متنی در طول این کتاب استفاده شده است. `CodeInText`: کلمات کد را در متن ، نام جدول پایگاه داده ، نام پوشش ها ، نام پرونده ها ، پسوند پرونده ها ، نام های مسیر ، URL های ساختگی ، ورودی کاربر و دسته های توییتر نشان می دهد. در اینجا مثالی وجود دارد: "شما می توانید محیط خود را در هر زمان با فرمان غیرفعال کردن غیرفعال کنید." بلوک کد به شرح زیر است:

```
from django.contrib import admin
from .models import Post

admin.site.register(Post)
```

وقتی می خواهیم توجه شما را به قسمت خاصی از یک کد جلب کنیم بلوک ، خطوط یا موارد مربوط به صورت زیر تنظیم می شوند:

```
INSTALLED_APPS = [  
    'django.contrib.admin',  
    'django.contrib.auth',  
    'django.contrib.contenttypes',  
    'django.contrib.sessions',  
    'django.contrib.messages',  
    'django.contrib.staticfiles',  
    'blog.apps.BlogConfig',  
]
```

هر ورودی یا خروجی خط فرمان به شرح زیر نوشته شده است:

```
$ python manage.py startapp blog
```

: اصطلاح جدید ، یک کلمه مهم یا کلماتی را که روی صفحه می بینید ، نشان می دهد. به عنوان مثال ، کلمات در منوها یا کادرهای گفتگو در متن مانند این ظاهر می شوند. در اینجا مثالی آورده شده است: "فرم را پر کرده و بر روی دکمه **SAVE** کلیک کنید."

Warnings or important notes appear like this.

Tips and tricks appear like this.

فصل اول

ایجاد یک برنامه وبلاگ

در این کتاب نحوه ساخت پروژه های کامل جنگو، آماده برای استفاده در تولید یاد خواهید گرفت. در صورتی که هنوز Django را نصب نکرده باشید، یاد می گیرید که چگونه این کار را در قسمت اول این فصل انجام دهید. این فصل نحوه ایجاد یک برنامه ساده وبلاگ با استفاده از جنگو را در بر می گیرد. هدف از این فصل، یک ایده کلی در مورد چگونگی عملکرد چارچوب، درک چگونگی تعامل مؤلفه های مختلف با یکدیگر و درک مهارت های لازم برای ایجاد پروژه های جنگو با قابلیت های اساسی در اختیار شماست. شما بدون ایجاد جزئیات در مورد جزئیات، از طریق ایجاد یک پروژه کامل راهنمایی خواهید شد. اجزای مختلف چارچوب در این کتاب به تفصیل شرح داده خواهد شد.

این فصل موضوعات زیر را شامل می شود:

نصب جنگو و ایجاد اولین پروژه خود

طراحی مدل ها و ایجاد مهاجرت مدل

ایجاد یک سایت مدیریت برای مدل های خود

همکاری با QuerySet و مدیران

ایجاد نماها، قالب ها و URL ها

افزودن صفحه نمایش به لیست نمایش با استفاده از نماهای مبتنی بر کلاس Django

نصب جنگو

اگر Django را قبلًا نصب کرده اید ، می توانید از این بخش پرس کرده و مستقیماً به قسمت ایجاد اولین بخش پروژه خود بروید. جنگو به عنوان پکیج پایتون عرضه می شود و بنابراین می تواند در هر محیط پایتون نصب شود. اگر هنوز Django را نصب نکرده اید ، راهنمای سریع برای نصب Django برای توسعه محلی است.

اگر Django 2.0 به نسخه پایتون 3.4 یا بالاتر نیاز دارد. در مثالهای این کتاب از پایتون 3.6.5 استفاده خواهیم کرد. اگر از لینوکس یا macOS استفاده می کنید ، احتمالاً پایتون را نصب کرده اید. اگر از Windows استفاده می کنید ، می توانید پک نصب کننده Python را در <https://www.python.org/downloads/windows> بارگیری کنید.

اگر مطمئن نیستید که Python روی رایانه شما نصب شده است ، می توانید با تایپ کردن پایتون در پوسته ، آن را تأیید کنید. اگر چیزی شبیه به موارد زیر می بینید ، پایتون روی رایانه شما نصب می شود:

```
Python 3.6.5 (v3.6.5:f59c0932b4, Mar 28 2018, 03:03:55)
[GCC 4.2.1 (Apple Inc. build 5666) (dot 3)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

اگر نسخه نصب شده Python شما از 3.4 پایین تر است ، یا اگر Python روی رایانه شما نصب نشده است ، Python 3.6.5 را از <https://www.python.org/downloads/> بارگیری کنید و نصب کنید.

از آنجا که از پایتون 3 استفاده خواهید کرد ، نیازی به نصب بانک اطلاعاتی ندارید. این نسخه Python همراه با یک پایگاه داده داخلی SQLite است. یک پایگاه داده سبک است که می توانید با توسعه مجدد Django از آن استفاده کنید.

اگر می خواهید برنامه خود را در یک محیط تولید مستقر کنید ، باید از یک بانک اطلاعاتی پیشرفته مانند PostgreSQL ، MySQL یا Oracle استفاده کنید. می توانید اطلاعات بیشتری در مورد نحوه اجرای بانک اطلاعاتی خود با Django در <https://docs.djangoproject.com/en/2.0/topics/install/#database-installation> دریافت کنید.

ایجاد یک محیط جدا شده پایتون

توصیه می شود برای ایجاد محیط های جدا شده Python از virtualenv استفاده کنید تا بتوانید از نسخه های مختلف بسته استفاده کنید

پروژه های مختلف که بسیار عملی تر از نصب پایتون هستند

بسته های مختلف سیستم: یکی دیگر از مزیت های استفاده از virtualenv این است

برای نصب بسته های پایتون به هیچ امتیاز مدیریتی احتیاج نخواهید داشت. برای نصب دستور زیر را در پوسته خود اجرا کنید

```
pip install virtualenv
```

پس از نصب virtualenv ، یک محیط جدا شده با دستور زیر ایجاد کنید:

```
virtualenv my_env
```

با استفاده از محیط Python شما یک my_env / فهرست ایجاد می کنید. هر کتابخانه Python که نصب می کنید در حالی که محیط مجازی شما فعال است به فهرست my_env / lib / python3.6 / site-pack ها می رود.

می توانید مسیری را که پایتون 3 نصب شده است بیابید و از آن برای ایجاد محیط مجازی با دستورات زیر استفاده کنید:

```
zenx$ which python3  
/Library/Frameworks/Python.framework/Versions/3.6/bin/python3  
zenx$ virtualenv my_env -p
```

```
/Library/Frameworks/Python.framework/Versions/3.6/bin/python3
```

برای فعال کردن محیط مجازی ، دستور زیر را اجرا کنید:

```
source my_env/bin/activate
```

خط پوسته نام محیط مجازی فعال محصور در پرانتز را به شرح زیر خواهد داشت:

```
(my_env)laptop:~ zenx$
```

می توانید اطلاعات بیشتر در مورد virtualenv را در اینجا پیدا کنید

<https://virtualenv.pypa.io/fa/latest>

در بالای virtualenv ، می توانید از بسته بندی virtualenv استفاده کنید. این ابزار بسته بندی هایی را فراهم می کند که ایجاد و مدیریت محیط های مجازی شما را آسان تر می کند. می توانید آن را از

<https://virtualenvwrapper.readthedocs.io/fa/latest>

بارگیری کنید.

نصب جنگو با پیپ

سیستم مدیریت بسته بندی پیپ روش مناسب برای نصب مجدد جنگو است. Python 3.6 همراه با پیپ از پیش نصب شده است ، اما می توانید دستورالعمل نصب پیپ را در <https://pip.pypa.io/en/stable/installing> پیدا کنید. دستور زیر را در قسمت پوسته نصب کنید تا Django را با pip نصب کنید:

```
pip install Django
```

جنگو در بسته های / فهرست دایرکتوری محیط مجازی شما در سایت Python نصب خواهد شد.

حال بررسی کنید که Django با موفقیت نصب شده است یا خیر. پایتون را روی یک ترمینال اجرا کنید ، جنگو را وارد کنید و نسخه آن را به شرح زیر بررسی کنید:

```
>>> import django  
>>> django.get_version()
```

ایجاد اولین پروژه خود

اولین پروژه جنگو ما ساخت و بلاگ کاملی خواهد بود. Django دستوری را فراهم می کند که به شما امکان می دهد ساختار اولیه فایل پروژه را ایجاد کنید. دستور زیر را از پوسته خود اجرا کنید:

```
django-admin startproject mysite
```

این کار یک پروژه جنگو با نام سایت من ایجاد می کند.

برای جلوگیری از درگیری از نامگذاری پروژه ها با مازول های پایتون یا جنگو اجتناب کنید.

بیایید نگاهی به ساختار پروژه ایجاد کنیم:

```
mysite/  
  manage.py  
mysite/  
  __init__.py  
  settings.py  
  urls.py  
  wsgi.py
```

این پرونده ها به شرح زیر است:

: این یک ابزار خط فرمان است که برای تعامل با پروژه شما استفاده می شود. یک بند نازک در اطراف management.py است. شما نیازی به ویرایش این پرونده ندارید.

/ : این فهرست راهنمای پروژه شما است که از پرونده های زیر تشکیل شده است:

.py__init__: پرونده ای خالی که به پایتون می گوید دایرکتوری mysite را به عنوان یک مازول پایتون درمان کنید.

: settings.py این تنظیمات و پیکربندی پروژه شما را نشان می دهد و حاوی تنظیمات پیش فرض اولیه است.

: urls.py این مکانی است که الگوهای URL شما در آن زندگی می کند. هر URL که در اینجا تعریف شده است ، برای نمایش نقشه برداری می شود.

: wsgi.py این پیکربندی برای اجرای پروژه شما به عنوان یک برنامه وب سرور (WSGI) GatewayInterface است.

پرونده settings.py تولید شده شامل تنظیمات پروژه است ، از جمله تنظیمات اولیه برای استفاده از پایگاه داده SQLite 3 و لیستی به نام INSTALLED_APPS ، که شامل برنامه های متداول جنگو است که بصورت پیش فرض به پروژه شما اضافه می شوند. ما این برنامه ها را بعداً در بخش تنظیمات پروژه خواهیم گذراند.

برای تکمیل راه اندازی پروژه ، ما نیاز به ایجاد جداول در بانک اطلاعاتی مورد نیاز برنامه های ذکر شده در INSTALLED_APPS داریم. پوسته را باز کرده و دستورات زیر را اجرا کنید:

```
cd mysite
python manage.py migrate
```

خروجی را با خطوط زیر به پایان می رسانید:

```
Applying contenttypes.0001_initial... OK
Applying auth.0001_initial... OK
Applying admin.0001_initial... OK
Applying admin.0002_logentry_remove_auto_add... OK
Applying contenttypes.0002_remove_content_type_name... OK
Applying auth.0002_alter_permission_name_max_length... OK
Applying auth.0003_alter_user_email_max_length... OK
Applying auth.0004_alter_user_username_opts... OK
Applying auth.0005_alter_user_last_login_null... OK
Applying auth.0006_require_contenttypes_0002... OK
Applying auth.0007_alter_validators_add_error_messages... OK
```

```
Applying auth.0008_alter_user_username_max_length... OK
Applying auth.0009_alter_user_last_name_max_length... OK
Applying sessions.0001_initial... OK
```

اجرای سرور توسعه

Django به همراه یک وب سرور سبک وزن است تا بتواند سریع کد شما را اجرا کند ، بدون اینکه نیازی به صرف وقت در تنظیم سرور تولید باشد. هنگامی که سرور توسعه Django را اجرا می کنید ، کنترل تغییرات در کد شما را کنترل می کند. این خودکار بازگیری مجدد می شود و شما را از بازگیری مجدد دستی پس از تغییر کد آزاد می کند. با این وجود ممکن است متوجه برخی اقدامات مانند اضافه کردن پرونده های جدید به پروژه خود نشوید ، بنابراین مجبور خواهید بود در این موارد سرور را مجدداً راه اندازی کنید.

با وارد کردن دستور زیر از پوشه root پروژه ، سرور توسعه را شروع کنید:

```
python manage.py runserver
```

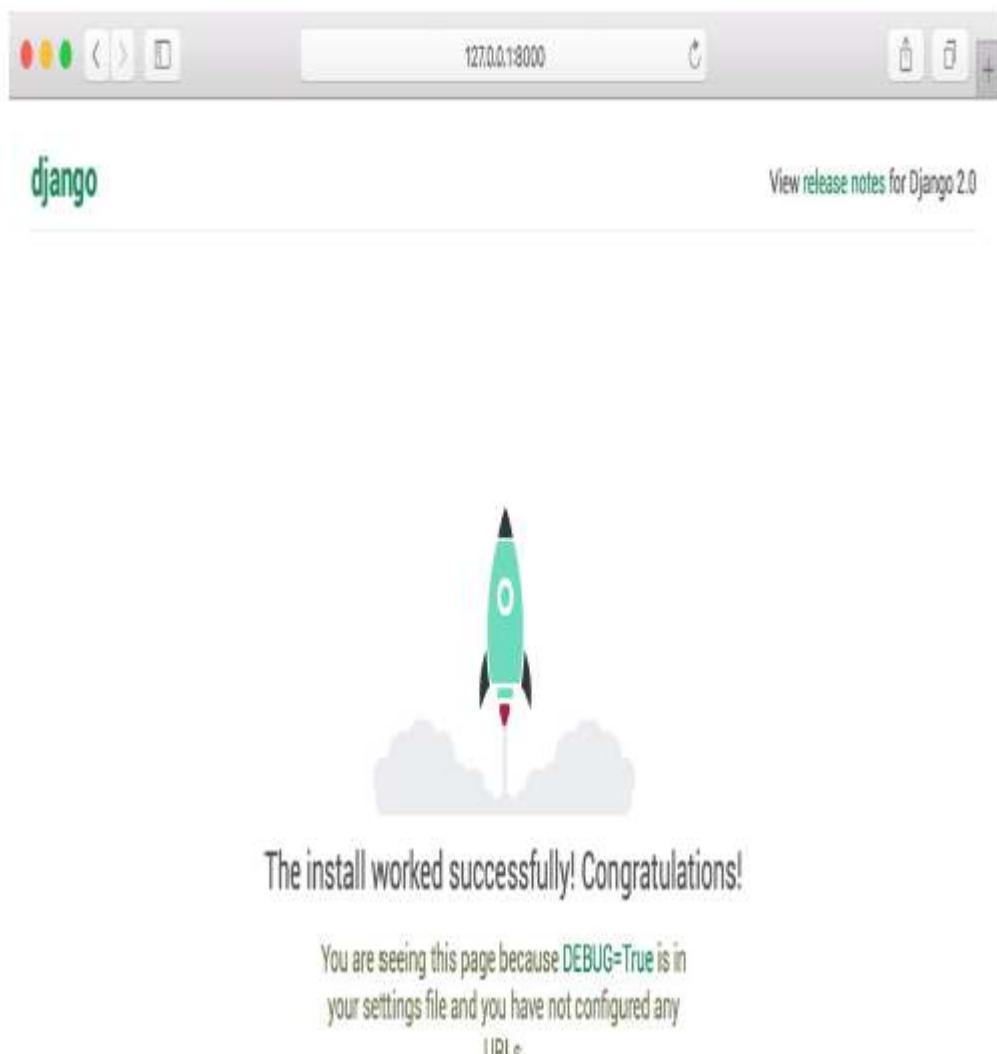
شما باید چیزی شبیه به این را ببینید:

```
Performing system checks...

System check identified no issues (0 silenced).
May 06, 2018 - 17:17:31
Django version 2.0.5, using settings 'mysite.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CONTROL-C.
```

اکنون در مرورگر خود 127.0.0.1:8000 را باز کنید.

شما باید صفحه ای را مشاهده کنید که بیان می کند پروژه با موفقیت در حال اجرا است همانطور که در تصویر زیر نشان داده شده است:



تصویر قبلی نشان می دهد که Django در حال اجرا است. اگر نگاهی به کنسول خود بیندازید ، درخواست GET را که توسط مرورگر شما انجام شده است مشاهده خواهید کرد:

```
[06/May/2018 17:20:30] "GET / HTTP/1.1" 200 16348
```

هر درخواست HTTP توسط سرور توسعه در کنسول وارد می شود. هر خطایی که هنگام اجرای سرور توسعه رخ دهد ، در کنسول نیز ظاهر می شود.

شما می توانید Django را برای اجرای سرور توسعه بر روی هاست و پورت دلخواه نشان دهید یا به آن بگویید که پروژه خود را بازگیری یک پرونده تنظیمات مختلف ، به شرح زیر انجام دهد:

```
python manage.py runserver 127.0.0.1:8001 \
--settings=mysite.settings
```

به یاد داشته باشید که این سرور فقط برای توسعه در نظر گرفته شده است و برای استفاده در تولید مناسب نیست. برای استقرار Django در یک محیط تولید ، شما باید آن را به عنوان یک برنامه WSGI با استفاده از یک سرور وب واقعی ، مانند Gunicorn ، Apache یا WSGI اجرا کنید. می توانید اطلاعات بیشتری در مورد نحوه استقرار Django با سرورهای مختلف وب را در <https://docs.djangoproject.com/en/2.0/howto/deployment/wsgi/> کسب کنید.

فصل سیزدهم ، چگونگی راه اندازی یک محیط تولید برای پروژه های جنگو را توضیح می دهد.

تنظیمات پروژه

باید پرونده تنظیمات را باز کنیم و به پیکربندی پروژه خود نگاهی بیندازیم. تنظیمات مختلف وجود دارد که Django در این پرونده شامل می شود ، اما اینها فقط بخشی از کلیه تنظیمات Django موجود است.

می توانید تمام تنظیمات و مقادیر پیش فرض آنها را در داخل مشاهده کنید

[./https://docs.djangoproject.com/fa/2.0/ref/settings](https://docs.djangoproject.com/fa/2.0/ref/settings)

تنظیمات زیر قابل مشاهده است:

DEBUG بولی است که حالت اشکال زدایی پروژه را روشن یا خاموش می کند. اگر روی True تنظیم شده باشد ، هنگامی که یک استثناء ناخواسته توسط برنامه شما ایجاد می شود ، جنگو صفحات خطأ را نمایش می دهد. وقتی به یک محیط تولید می روید ، به یاد داشته باشید که باید آن را False تنظیم کنید. هرگز روشن نکنید که سایتی روشن و روشن باشد زیرا داده های حساس مربوط به پروژه را در معرض دید سایت قرار می دهید.

ALLOWED_HOSTS در حالی که حالت اشکال زدایی یا زمانی که آزمایش ها انجام شده است اعمال نمی شود. هنگامی که سایت خود را به سمت تولید سوق دهید و DEBUG را به False تنظیم کنید ، باید دامنه / هاست خود را به این تنظیمات اضافه کنید تا بتوانید به آن اجازه دهید سایت Django شما را ارائه دهد.

INSTALLED_APPS تنظیماتی است که باید برای همه پروژه ها ویرایش کنید.

این تنظیمات به Django می گوید که چه برنامه هایی برای این سایت فعال هستند. به طور پیش فرض ، جنگو شامل برنامه های زیر است:

django.contrib.admin: یک سایت مدیریت

django.contrib.auth: چارچوب احراز هویت

django.contrib.contenttypes: چارچوبی برای کنترل انواع محتوا

django.contrib.sessions: چارچوب جلسه

django.contrib.messages: یک چارچوب پیام رسانی

django.contrib.staticfiles: چارچوبی برای مدیریت پرونده های استاتیک

MIDDLEWARE لیستی است که حاوی واسطه برای اجرا است.

ROOT_URLCONF مازول پایتون را نشان می دهد که URL های اصلی برنامه شما تعریف شده است.

DATABASES فرهنگ لغت است که شامل تنظیمات کلیه بانکهای اطلاعاتی است که در پروژه استفاده می شود. همیشه باید یک بانک اطلاعاتی پیش فرض وجود داشته باشد. پیکربندی پیش فرض از SQLite3database استفاده می کند.

LANGUAGE_CODE کد زبان پیش فرض را برای این سایت Django تعریف می کند.

USE_TZ به Django می گوید پشتیبانی از منطقه زمانی را فعال یا غیرفعال کنید.

DateTime آگاه به منطقه زمانی همراه است. این تنظیم هنگامی که یک پروژه جدید را با استفاده از دستور مدیریت شروع پروژه ایجاد می کنید ، روی True تنظیم می شود.

اگر چیزهای زیادی را که می بینید نفهمید نگران نباشید. شما می توانید تنظیمات مختلف جنگو را در فصل های بعدی بیاموزید.

پروژه ها و برنامه های کاربردی

در طول این کتاب ، بارها و بارها با اصطلاحات پروژه و برنامه مواجه می شوید. در جنگو ، یک پروژه با نصب برخی از تنظیمات ، جنگو محسوب می شود. برنامه گروهی از مدل ها ، ناماها ، قالب ها و URL ها است. برنامه های کاربردی برای ارائه برخی ویژگی های خاص با چارچوب تعامل دارند و ممکن است در پروژه های مختلف مورد استفاده مجدد قرار بگیرند. شما می توانید این پروژه را به عنوان وب سایت خود ، که شامل چندین برنامه مانند وبلاگ ، ویکی یا انجمن است فکر کنید ، که می تواند توسط سایر پروژه ها نیز مورد استفاده قرار گیرد.

ایجاد یک برنامه

حال ، بیایید اولین برنامه Django را ایجاد کنیم. ما از ابتدا یک برنامه وبلاگ ایجاد خواهیم کرد. از فهرست اصلی پروژه ، دستور زیر را اجرا کنید:

```
python manage.py startapp blog
```

این ساختار اصلی برنامه را ایجاد می کند ، که اینطور به نظر می رسد:

```
blog/
__init__.py
admin.py
apps.py
migrations/
__init__.py
models.py
tests.py
views.py
```

این پرونده ها به شرح زیر است:

admin.py: اینجاست که شما مدل هایی را ثبت می کنید تا آنها را در سایت مدیریت جنگو قرار دهید — استفاده از سایت مدیر جنگو اختیاری است.

apps.py: این شامل تنظیمات اصلی برنامه وبلاگ است.

migrations: این فهرست شامل مهاجرت بانک اطلاعاتی برنامه شما خواهد بود. مهاجرت به Django اجازه می دهد تا تغییرات مدل شما را ردیابی کند و مطابق با آن بانک اطلاعات را همگام کند.

model.py: مدل های داده برنامه شما — همه برنامه های Django نیاز به یک فایل model.py دارند ، اما این پرونده ممکن است خالی بماند.

tests.py: اینجاست که می توانید تست هایی را برای برنامه خود اضافه کنید.

views.py: منطق برنامه شما به اینجا می رود. هر شاخه درخواست HTTP را دریافت می کند ، آن را پردازش می کند و پاسخی را بر می گرداند.

طراحی شمای داده های وبلاگ

ما با تعریف مدل های داده برای مدل وبلاگ خود ، طراحی شمای داده های بلاگ خود را شروع خواهیم کرد ، یک کلاس پایتون است که زیر کلاس ها

django.db.models.Modeling که در آن هر ویژگی یک زمینه پایگاه داده را نشان می دهد. جنگو برای هر مدلی که در پرونده model.py تعریف شده است ، جدول ایجاد می کند. هنگامی که شما یک مدل را ایجاد می کنید ، Django یک API عملی برای شما فراهم می کند تا بتوانید اشیاء موجود در بانک اطلاعات را به راحتی جستجو کنید.

ابتدا یک مدل Post تعریف خواهیم کرد. خطوط زیر را به پرونده models.py برنامه وبلاگ اضافه کنید:

```

from django.db import models
from django.utils import timezone
from django.contrib.auth.models import User

class Post(models.Model):
    STATUS_CHOICES = (
        ('draft', 'Draft'),
        ('published', 'Published'),
    )
    title = models.CharField(max_length=250)
    slug = models.SlugField(max_length=250,
                           unique_for_date='publish')
    author = models.ForeignKey(User,
                               on_delete=models.CASCADE,
                               related_name='blog_posts')
    body = models.TextField()
    publish = models.DateTimeField(default=timezone.now)
    created = models.DateTimeField(auto_now_add=True)
    updated = models.DateTimeField(auto_now=True)
    status = models.CharField(max_length=10,
                             choices=STATUS_CHOICES,
                             default='draft')

    class Meta:
        ordering = ('-publish',)

```

```

def __str__(self):
    return self.title

```

این مدل داده های ما برای پست های وبلاگ است. بیایید نگاهی به زمینه هایی که برای این مدل تعریف کرده ایم:

عنوان: این قسمت برای عنوان پست است. این قسمت CharField است که در پایگاه داده SQL به یک ستون VARCHAR ترجمه می شود.

slug: این فیلدی است که در URL ها استفاده می شود. Slug یک برچسب کوتاه است که فقط شامل حروف ، اعداد ، علامت های زیر یا امتیازات است. ما از زمینه slug برای ساختن URL های زیبا و دوستانه برای SEO برای پست های وبلاگ خود استفاده خواهیم کرد. ما پارامتر unique_for_date را به این قسمت اضافه کرده ایم تا بتوانیم با استفاده از تاریخ

انتشار و slug آنها نشانی اینترنتی برای پست ها ایجاد کنیم. جنگو مانع از آن خواهد شد که چندین پست در همان تاریخ دارای همان سورتمه ای مشابه باشد.

نویسنده: این زمینه یک کلید خارجی است. این یک رابطه چند به یک را تعریف می کند. ما به جنگویی گوییم که هر پست توسط یک کاربر نوشته شده است و یک کاربری تواند هر تعداد ارسال را بنویسد. برای این زمینه ، جنگو با استفاده از کلید اصلی مدل مربوطه ، یک کلید خارجی را در بانک اطلاعات ایجاد می کند. در این حالت ، ما به مدل کاربر سیستم احراز هویت Django تکیه می کنیم. پارامتر on_delete رفتاری را که هنگام حذف شیء ارجاع شده ، اتخاذ می کند ، مشخص می کند. این مختص جنگو نیست؛ این یک استاندارد SQL است. با استفاده از CASCADE ، مشخص می کنیم که وقتی کاربر مرجع حذف می شود ، پایگاه داده نیز پست های مربوط به وبلاگ خود را حذف می کند. می توانید به همه گزینه های ممکن در <https://docs.djangoproject.com/fa/2> نگاهی بیندازید.

ما نام رابطه معکوس را از کاربر به پست با ویژگی related_name مشخص می کنیم. این به ما امکان دسترسی آسان به اشیاء مرتبط را می دهد. بعداً در این مورد بیشتر خواهیم آموخت.

body: این بدن ارسال است. این قسمت یک فیلد متنی است که در پایگاه داده SQL به یک ستون TEXT ترجمه می شود.
انتشار: این DateTime زمان انتشار پست را نشان می دهد.

ما از متدهای زمانی Django به عنوان مقدار پیش فرض استفاده می کنیم. این تاریخ فعلی DateTime را با فرمت آگاهی از منطقه زمانی بری گرداند. شما می توانید از آن به عنوان یک نسخه آگاه از منطقه زمانی آگاهانه استاندارد Python استفاده کنید. روش ایجاد شده: این DateTime نشانگر زمان ایجاد پست است.

از آنجا که ما در اینجا از auto_now_add استفاده می کنیم ، هنگام ایجاد یک شیء ، تاریخ به طور خودکار ذخیره می شود.
status: این قسمت وضعیت یک پست را نشان می دهد. ما از یک پارامتر انتخاب استفاده می کنیم ، بنابراین می توان مقدار این فیلد را فقط روی یکی از گزینه های داده شده تنظیم کرد.

جنگو با انواع مختلفی از زمینه ها ارائه می شود که می توانید از آنها برای تعریف مدل های خود استفاده کنید. می توانید انواع فیلدها را در <https://docs.djangoproject.com/en/2.0/ref/models/fields/> پیدا کنید.

کلاس متأدخل مدل شامل ابرداده است. ما وقتی به جستجوی پایگاه داده می پردازیم ، به Django می خواهیم نتایج را در قسمت نشر به ترتیب نزولی مرتب کند. ترتیب نزولی را با استفاده از پیشوند منفی مشخص می کنیم. با این کار ، پستهای منتشر شده اخیراً اول ظاهر می شوند

روش str نمایش پیش فرض قابل خواندن است. جنگو از آن در بسیاری از مکان ها مانند سایت مدیریت استفاده خواهد کرد.

برنامه خود را فعال کنید

به منظور اینکه جنگو برنامه خود را پیگیری کند و بتواند جداول بانک اطلاعاتی را برای مدل های خود ایجاد کند ، باید آن را فعال کنیم. برای انجام این کار ، پرونده تنظیمات را ویرایش کنید و یک blog.apps.BlogConfig را به تنظیمات INSTALLED_APPS اضافه کنید. می بایست شبیه به این باشد:

```
INSTALLED_APPS = [  
    'django.contrib.admin',  
    'django.contrib.auth',  
    'django.contrib.contenttypes',  
    'django.contrib.sessions',  
    'django.contrib.messages',  
    'django.contrib.staticfiles',  
    'blog.apps.BlogConfig',  
]
```

کلاس BlogConfig پیکربندی برنامه شما است. اکنون جنگو می داند که برنامه ما برای این پروژه فعال است و قادر به پارگیری مدل های آن خواهد بود.

ایجاد و اعمال مهاجرت

اکنون که ما یک مدل داده برای پست های وبلاگ خود داریم ، برای آن به یک جدول دیتابیس احتیاج داریم. جنگو با یک سیستم مهاجرت همراه است که تغییرات انجام شده در مدل ها را ردیابی می کند و به آنها امکان می دهد تا آنها را به پایگاه داده منتقل کنند. فرمان مهاجرت را برای همه برنامه های ذکر شده در INSTALLED_APPS اعمال می کند. این پایگاه داده را با مدل های فعلی و مهاجرت های موجود همزمان می کند.

در مرحله اول ، شما نیاز به ایجاد یک مهاجرت اولیه برای مدل Post خود دارید.

در فهرست اصلی پروژه خود ، دستور زیر را اجرا کنید:

```
python manage.py makemigrations blog
```

شما باید خروجی زیر را بدست آورید:

```
Migrations for 'blog':  
  blog/migrations/0001_initial.py  
    - Create model Post
```

به تازگی پرونده initial.py_0001 را در پوشه مهاجرت برنامه و بلاگ ایجاد کرد. می توانید آن پرونده را باز کنید تا ببینید چگونه یک مهاجرت ظاهر می شود. یک مهاجرت وابستگی به سایر مهاجرت ها و عملیات انجام شده در پایگاه داده را برای همگام سازی آن با تعییرات مدل مشخص می کند.

باید نگاهی به کد SQL بیندازیم که Django در بانک اطلاعاتی آن را اجراء کند تا جدول برای مدل ما ایجاد شود. فرمان مهاجرت SQL نام های مهاجرت را می گیرد و SQL آنها را بدون اجرای آن بر می گرداند. دستور زیر را برای بررسی بازده SQL در اولین مهاجرت ما اجرا کنید:

```
python manage.py sqlmigrate blog 0001
```

خروجی باید به شرح زیر باشد:

```
BEGIN;  
--  
-- Create model Post  
--  
CREATE TABLE "blog_post" ("id" integer NOT NULL PRIMARY KEY AUTOINCREMENT,  
"title" varchar(250) NOT NULL, "slug" varchar(250) NOT NULL, "body" text NOT  
NULL, "publish" datetime NOT NULL, "created" datetime NOT NULL, "updated"  
datetime NOT NULL, "status" varchar(10) NOT NULL, "author_id" integer NOT  
NULL REFERENCES "auth_user" ("id"));  
CREATE INDEX "blog_post_slug_b95473f2" ON "blog_post" ("slug");  
CREATE INDEX "blog_post_author_id_dd7a8485" ON "blog_post" ("author_id");  
COMMIT;
```

خروجی دقیق به دیتابیس مورد استفاده شما بستگی دارد. خروجی قبلی برای SQLite تولید می شود. همانطور که در خروجی قبلی مشاهده می کنید ، Django با ترکیب نام برنامه و نام کوچک مدل (blog_post) نام جدول ها را ایجاد می کند ، اما همچنان می توانید با استفاده از db_tableattribute. Django یک کلید اصلی را به طور خودکار ایجاد کند ، اما می توانید با مشخص کردن primary_key=True در یکی از زمینه های مدل خود ، این را نادیده بگیرید. کلید اصلی پیش فرض یک ستون id است که از یک عدد صحیح تشکیل شده است که به صورت خودکار افزایش می یابد. این ستون مربوط به قسمت id است که به طور خودکار به مدل های شما اضافه می شود. بیایید پایگاه داده خود را با مدل جدید همگام سازی کنیم. دستور زیر را برای اعمال مهاجرت های موجود اجرا کنید:

```
python manage.py migrate
```

خروجی که دریافت می کنید با خط زیر پایان می یابد:

```
Applying blog.0001_initial... OK
```

ما فقط برای برنامه های ذکر شده در INSTALLED_APPS ، از جمله برنامه وبلاگ خود ، مهاجرت را اعمال کردیم. پس از اعمال مهاجرت ، بانک اطلاعاتی وضعیت فعلی مدل های ما را نشان می دهد. اگر فایل model.py خود را ویرایش می کنید تا زمینه های مدل های موجود اضافه یا حذف شود ، یا در صورت اضافه کردن مدل های جدید ، مجبور شوید با استفاده از دستور make migrations یک مهاجرت جدید ایجاد کنید. مهاجرت به Django اجازه می دهد تا تغییرات مدل را پیگیری کند. سپس ، شما باید آن را با دستور migrate اعمال کنید تا پایگاه داده همگام با مدل های خود حفظ شود.

ایجاد یک سایت مدیریت برای مدل های خود

اکنون که مدل Post را تعریف کردیم ، یک سایت ساده برای مدیریت پست های وبلاگ شما ایجاد خواهیم کرد. جنگو با یک رابط کاربری داخلی ساخته شده است که برای ویرایش محتوا بسیار مفید است. سایت مدیر سایت جنگو با خواندن ابرداده مدل شما و ارائه یک رابط آماده تولید برای ویرایش محتوا بصورت پویا ساخته شده است. شما می توانید آن را از جعبه استفاده کنید ، نحوه تنظیم مدل های شما را در آن پیکربندی می کند.

برنامه django.contrib.admin گنجانده شده است ، بنابراین نیازی به اضافه کردن آن نداریم.

ایجاد یک مدیر

در مرحله اول ، برای مدیریت سایت مدیریت باید یک کاربر ایجاد کنیم. دستور زیر را اجرا کنید:

```
python manage.py createsuperuser
```

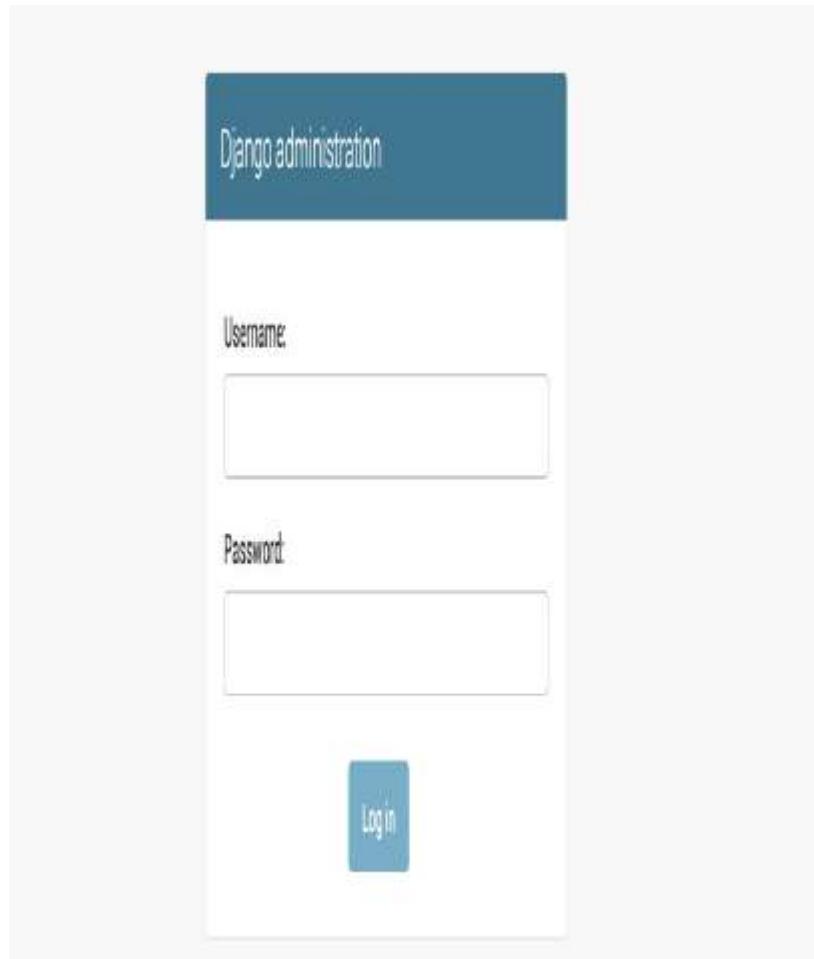
خروجی زیر را مشاهده خواهید کرد. نام کاربری ، ایمیل و رمز عبور مورد نظر خود را به شرح زیر وارد کنید:

```
Username (leave blank to use 'admin'): admin
Email address: admin@admin.com
Password: *****
Password (again): *****
Superuser created successfully.
```

سایت مدیریت جنگو

اکنون ، سرور توسعه را با دستور `python management.py run server` شروع کرده و `/http://127.0.0.1:8000/admin` را در مرورگر خود باز کنید.

همانطور که در تصویر زیر نشان داده شده است ، باید صفحه ورود به سیستم مدیریت را مشاهده کنید:



با استفاده از اعتبار کاربری که در مرحله ایجاد کرده اید ، وارد سیستم شوید. همانطور که در صفحه مشاهده می کنید ، صفحه فهرست سایت سرپرست را مشاهده خواهید کرد

تصویر زیر:

The screenshot shows the Django administration interface. At the top, it says "Django administration" and "WELCOME, ADMIN. VIEW SITE / CHANGE PASSWORD / LOG OUT". Below this, the title "Site administration" is displayed. On the left, there is a sidebar titled "AUTHENTICATION AND AUTHORIZATION" containing links for "Groups" and "Users", each with "Add" and "Change" buttons. To the right, there are two sections: "Recent actions" (empty) and "My actions" (also empty, with a message "None available").

مدلهای Group و User که در عکس قبلی مشاهده می کنید بخشی از چارچوب تأیید هویت Django است که در indjango.contrib.auth واقع شده است. اگر روی Users کلیک کنید ، کاربری را که قبلاً ایجاد کرده اید خواهد دید. مدل ارسال برنامه و بلاگ شما با این مدل کاربر ارتباط دارد. به یاد داشته باشید که این رابطه ای است که توسط قسمت نویسنده تعریف شده است.

مدل های خود را به سایت مدیریت اضافه کنید

بیایید مدل های و بلاگ خود را به سایت اضافه کنیم. در پرونده admin.py برنامه و بلاگ خود را ویرایش کنید تا صورت زیر ظاهر شود:

```
from django.contrib import admin
from .models import Post

admin.site.register(Post)
```

اکنون ، سایت مدیر را در مرورگر خود بارگیری کنید. شما باید مدل Post خود را در سایت سربرست مشاهده کنید ، به شرح زیر:

Site administration

AUTHENTICATION AND AUTHORIZATION

Groups

 Add  Change

Users

 Add  Change

BLOG

Posts

 Add  Change

Recent actions

My actions

None available

این آسان بود ، درست است؟ هنگامی که شما یک مدل را در سایت مدیر Django ثبت می کنید ، با استفاده از مدل های درون خود ، یک رابط کاربر پسند ایجاد می کنید که به شما امکان می دهد مواردی را به روشی ساده لیست ، ویرایش ، ایجاد و حذف کنید.

برای افزودن پست جدید ، روی پیوند افزودن در کنار پستها کلیک کنید. به فرم ایجاد شده توسط Django که به صورت پویا برای مدل شما ایجاد شده است توجه داشته باشید همانطور که در تصویر زیر نشان داده شده است:

Home > Blog > Posts > Add post

Add post

Title:

Slug:

Author: +

Body:

Publish: Date: 2017-12-14 Today Time: 08:54:24 Now
Note: You are 3 hours ahead of server time.

Status: Draft ▼

جنگو برای هر نوع فیلد از ابزارک های مختلف استفاده می کند. حق زمینه های پیچیده ای مانند DateTimeField نیز به راحتی نمایش داده می شوند.

رابط یک انتخاب کننده تاریخ JavaScript است.

فرم را پر کرده و بر روی دکمه SAVE کلیک کنید. همانطور که در تصویر زیر نشان داده شده است ، باید با یک پیام موفق و پستی که تازه ایجاد کرده اید ، به صفحه لیست پست هدایت شوید:

 The post "Who was Django Reinhardt?" was added successfully.

Select post to change

ADD POST 

Action: Go 0 of 1 selected

POST

Who was Django Reinhardt?

1 post

سفارشی کردن نحوه نمایش مدلها

اکتون نگاهی خواهیم داشت به نحوه شخصی سازی سایت سرپرست. فایل admin.py برنامه و بلاگ خود را ویرایش کنید و آن را به شرح زیر تغییر دهید:

```
from django.contrib import admin
from .models import Post

@admin.register(Post)
class PostAdmin(admin.ModelAdmin):
    list_display = ('title', 'slug', 'author', 'publish',
                    'status')
```

ما به سایت مدیر جنگو می گوییم که مدل ما در آن ثبت شده است. سایت سرپرست با استفاده از یک کلاس سفارشی که از ModelAdmin به ارت می برد. در این کلاس می توانیم اطلاعاتی در مورد نحوه نمایش مدل در سایت سرپرست و نحوه تعامل با آن داشته باشیم. ویژگی list_display به شما امکان می دهد زمینه های مدل خود را که می خواهید در صفحه فهرست شیء مدیر نمایش دهید ، تنظیم کنید. تزئین کننده admin.register همان عملکردی را انجام می دهد که باید admin.site.register را برای جایگزینی با ثبت نام کلاس ModelAdmin که تزئین شده است ، جایگزین کنیم.

باید با استفاده از گزینه های مدیر مدل را با چند گزینه دیگر تنظیم کنیم، کد زیر:

```
@admin.register(Post)
class PostAdmin(admin.ModelAdmin):
    list_display = ('title', 'slug', 'author', 'publish',
                    'status')
    list_filter = ('status', 'created', 'publish', 'author')
    search_fields = ('title', 'body')
    prepopulated_fields = {'slug': ('title',)}
    raw_id_fields = ('author',)
    date_hierarchy = 'publish'

    ordering = ('status', 'publish')
```

به مرورگر خود برگردید و صفحه فهرست ارسال را پارگیری مجدد کنید. حالا ، اینگونه به نظر می رسد:

Select post to change

ADD POST



Search

/ 2017 December 14

Action



Go

0 of 1 selected



TITLE

SLUG

AUTHOR

PUBLISH

1 STATUS



Who was Django Reinhardt?

who-was-django-reinhardt

admin

Dec 14, 2017, 8:54 a.m.

Draft

1 post

FILTER

By status

All

Draft

Published

By created

Any date

Today

Past 7 days

This month

This year

By publish

Any date

Today

Past 7 days

This month

This year

می توانید مشاهده کنید که فیلدهای نمایش داده شده در صفحه لیست ارسال مواردی هستند که شما در ویژگی list_display مشخص کرده اید. در حال حاضر صفحه لیست نوار کناری سمت راست است که به شما امکان می دهد نتایج را توسط فیلدهای موجود در ویژگی list_filter فیلتر کنید. یک نوار جستجو در صفحه ظاهر شده است. دلیل این امر است که ما لیستی از فیلدهای جستجو را با استفاده از ویژگی search_fields تعریف کرده ایم. دقیقاً در زیر نوار جستجو، پیوندهای ناوبری برای پیمایش از طریق سلسله مراتب تاریخ وجود دارد: این با ویژگی date_hierarchy تعریف شده است. همچنین می توانید مشاهده کنید که ارسال ها توسط ستون های Status و Publish به صورت پیش فرض سفارش می شوند. ما سفارش پیش فرض را با استفاده از ویژگی سفارش گذاری مشخص کرده ایم. حال بر روی لینک Add سفارش می شوند.

کلیک کنید. در اینجا همچنین به برخی از تغییرات توجه خواهید کرد. همانطور که عنوان یک پست جدید را تایپ می کنید ، قسمت slug به صورت خودکار پر می شود. ما به Django گفتیم که با استفاده از ویژگی prepopulated_fields ، قسمت slug را با ورودی قسمت عنوان آماده کنید. همچنین ، اکنون ، قسمت نویسنده با ویجت جستجوی نمایش داده شده است که می تواند مقیاس بسیار بهتر از ورودی انتخابی کشویی هنگام داشتن هزاران کاربر همانطور که در تصویر زیر نشان داده شده است:



با چند خط کد ، نحوه نمایش مدل ما در سایت مدیر را سفارشی کرده ایم. روش های زیادی برای شخصی سازی و گسترش سایت مدیریت جنگو وجود دارد. در این مورد بعداً در این کتاب خواهید آموخت.

همکاری با QuerySet و مدیران

اکنون که یک سایت مدیریت کاملاً کاربردی برای مدیریت محتوای و بلاگ خود دارید ، وقت آن است که یاد بگیرید که چگونه می توانید اطلاعات را از پایگاه داده بازیابی کنید و با آن تعامل برقرار کنید. Django با یک API انتزاع بانک اطلاعاتی قادر تمند ارائه می شود که به شما امکان می دهد اشیاء را به راحتی ایجاد ، بازیابی ، به روزرسانی و حذف کنید. رابطه ی جنگو

MySQL با Mapper Oracle ، PostgreSQL و SQLite سازگار است. به یاد داشته باشید که می توانید پایگاه داده پروژه خود را در تنظیمات DATABASES پرونده تنظیمات.py پروژه خود تعریف کنید. Django می تواند همزمان با چندین پایگاه داده کار کند و شما می توانید برای ایجاد برنامه های مسیریابی سفارشی ، مسیریاب های پایگاه داده را برنامه ریزی کنید.

وقتی مدل داده های خود را ایجاد کردید ، Django یک API رایگان برای تعامل با آنها به شما می دهد. می توانید مرجع مدل داده های استناد رسمی را در اینجا پیدا کنید

[./https://docs.djangoproject.com/en/2.0/ref/models/](https://docs.djangoproject.com/en/2.0/ref/models/)

ایجاد اشیاء

ترمینال را باز کنید و دستور زیر را برای باز کردن پوسته پایتون اجرا کنید:

```
python manage.py shell
```

سپس خطوط زیر را تایپ کنید:

```
>>> from django.contrib.auth.models import User
>>> from blog.models import Post
>>> user = User.objects.get(username='admin')
>>> post = Post(title='Another post',
   slug='another-post',
   body='Post body.',
   author=user)
>>> post.save()
```

باید تجزیه و تحلیل آنچه این کد انجام می دهد را ببینیم. ابتدا شیء کاربری را با نام کاربری ادمین بازیابی می کنیم:

```
user = User.objects.get(username='admin')
```

متدهای get به شما امکان می دهد یک شیء واحد را از پایگاه داده بازیابی کنید. توجه داشته باشید که این روش نتیجه ای را می دهد که با پرس و جو مطابقت داشته باشد. اگر هیچ نتیجه ای توسط بانک اطلاعاتی بازگردانده نشود ، این روش یک استثناء MultipleObjectsReturned را ایجاد می کند ، و اگر پایگاه داده بیش از یک نتیجه را برگرداند ، یک استثناء DoesNotExist را جمع می کند. هر دو استثنای ویژگی های کلاس مدل است که پرس و جو در حال انجام است.

سپس ، ما یک نمونه پستی را با عنوان ، slug و body سفارشی ایجاد می کنیم و کاربری را که قبلاً به عنوان نویسنده بازیابی کرده بودیم تنظیم می کنیم.

```
post = Post(title='Another post', slug='another-post', body='Post body.',
author=user)
```

This object is in memory and is not persisted to the database.

سرانجام با استفاده از روش save شیء Post را در پایگاه داده ذخیره می کنیم:

```
post.save()
```

عملکرد قبلی یک عبارت INSERT SQL را در پشت صحنه انجام می دهد. ما دیدیم که چگونه ابتدایک شیء را در حافظه ایجاد کرده و سپس آن را به بانک اطلاعاتی ارجاع دهیم ، اما می توانیم با استفاده از روش `create()` ، در یک عمل واحد ، شیء را ایجاد کرده و آن را در بانک اطلاعاتی ادامه دهیم ، به شرح زیر:

```
Post.objects.create(title='One more post', slug='one-more-post', body='Post  
body.', author=user)
```

به روزرسانی اشیاء

اکنون عنوان پست را به چیز دیگری تغییر دهید و دوباره شی را ذخیره کنید:

```
>>> post.title = 'New title'  
>>> post.save()
```

این بار ، `save()` عبارت UPDATE SQL را انجام می دهد.

تغییراتی که شما در شی ایجاد می کنید تا زمانی که شما ذخیره نکنید ، در پایگاه داده باقی نمی ماند.

بازیابی اشیاء

نقشه نویسی رابطه-رابطه‌ی جنگو (ORM) بر اساس QuerySets است. QuerySet مجموعه‌ای از اشیاء از پایگاه داده شما است که می تواند چندین فیلتر برای محدود کردن نتایج داشته باشد. شما در حال حاضر می دانید که چگونه با استفاده از روش `get()` یک شی واحد از پایگاه داده را بازیابی کنید. ما با استفاده از `Post.objects.get()` به این روش دسترسی پیدا کرده ایم. هر مدل جنگو حداقل یک مدیر دارد و مدیر پیش فرض اشیاء نامیده می شوند. شما با استفاده از مدیر مدل خود یک شی دریافت می کنید. برای بازیابی همه اشیاء از جداول ، فقط از روش `all()` در مدیر پیش فرض اشیاء استفاده می کنید ، مانند این:

```
>>> all_posts = Post.objects.all()
```

به این ترتیب ما QuerySet را ایجاد می کنیم که تمام اشیاء موجود در پایگاه داده را باز می گرداند. توجه داشته باشید که این هنوز اجرایی نشده است.

Django QuerySets تنبل هستند؛ آنها فقط زمانی مورد ارزیابی قرار می گیرند که این رفتار QuerySet را بسیار کارآمد می کند. اگر QuerySet را به متغیر متغیر نرسانیم ، در عوض آن را مستقیماً روی پوسته پایتون می نویسیم ، عبارت SQL از اجرا می شود ، زیرا ما آن را مجبور می کنیم تا نتایج خروجی داشته باشد:

```
>>> Post.objects.all()
```

استفاده از روش فیلتر

برای فیلتر کردن QuerySet می توانید از روش filter() مدیر استفاده کنید. به عنوان مثال ، می توانیم تمام پستهای منتشر شده در سال 2017 را با استفاده از QuerySet زیر بازیابی کنیم:

```
Post.objects.filter(publish__year=2017)
```

همچنین می توانید با چند فیلد فیلتر کنید. به عنوان مثال ، می توانیم پستهای منتشر شده در سال 2017 توسط نویسنده را با نام کاربری admin بازیابی کنیم:

```
Post.objects.filter(publish__year=2017, author__username='admin')
```

این مساوی است با ساختن فیلترهای زنجیره ای همان QuerySet :

```
Post.objects.filter(publish__year=2017) \
    .filter(author__username='admin')
```

حذف

شما می توانید نتایج خاصی را از QuerySet خود با استفاده از `execute()` مدیر حذف کنید. به عنوان مثال ، می توانیم های منتشر شده در 2017 را بازیابی کنیم که عناوین آنها با چرا شروع نمی شود:

استفاده از `order_by`

شما می توانید با استفاده از متدهای `order_by` ، نتایج را با مختلفهای مختلف سفارش دهید. به عنوان مثال ، می توانید تمام اشیاء سفارش داده شده با عنوان آنها را به شرح زیر بازیابی کنید:

```
Post.objects.order_by('title')
```

نظم صعودی به این معنی است که شما می توانید ترتیب نزولی را با پیش شماره علامت منفی نشان دهید ، مانند این:

```
Post.objects.order_by('-title')
```

حذف اشیاء

اگر می خواهید یک شیء را حذف کنید ، می توانید با استفاده از `Delete()` این کار را از نمونه شی انجام دهید:

```
post = Post.objects.get(id=1)
post.delete()
```

وقتی QuerySets می شود

شما می توانید به همان اندازه فیلترهای دلخواه خود را با `QuerySet` به اشتراک بگذارید ، و تا زمانی که `QuerySet` ارزیابی نشود ، به پایگاه داده ضربه نخواهید یافت. `QuerySet` فقط در موارد زیر ارزیابی می شوند:

اولین باری که آنها را تکرار می کنید

به عنوان مثال ، وقتی آنها را برش می دهید ، `Post.objects.all()[3:]`

هنگامی که آنها را ترسی یا کش کردید

هنگامی که به آنها `len` یا `rep` می خوانید

وقتی صریحاً با آنها تماس می‌گیرید

وقتی آنها را در یک بیانیه آزمایش کنید، مانند `bool` ، `or` ، `and` ، یا `if`

Creating model managers

همانطور که قبلاً نیز اشاره کردیم، اشیاء مدیر پیش فرض هر مدلی است که کلیه اشیاء موجود در پایگاه داده را بازیابی می‌کند. با این حال، ما همچنین می‌توانیم مدیران سفارشی را برای مدل‌های خود تعریف کنیم. ما یک مدیر سفارشی برای بازیابی کلیه پست‌ها با وضعیت منتشر شده ایجاد خواهیم کرد.

دو روش برای اضافه کردن مدیران به مدل‌های شما وجود دارد: می‌توانید روش‌های مدیر اضافی را اضافه کنید یا `QuerySets` مدیر اولیه را تغییر دهید. روش اول API `Post.objects.my_manager` را در اختیار شما قرار می‌دهد و روش دوم `Post.my_manager.all` را برای شما فراهم می‌کند. مدیر به ما اجازه می‌دهد تا پست‌ها را با استفاده از `model.py` برنامه و بلاگ خود را ویرایش کنیم. برای افزودن مدیر سفارشی، فایل `model.py` بازیابی کنید:

```
class PublishedManager(models.Manager):
    def get_queryset(self):
        return super(PublishedManager,
                    self).get_queryset()\
            .filter(status='published')

class Post(models.Model):
    # ...
    objects = models.Manager() # The default manager.
    published = PublishedManager() # Our custom manager.
```

متدهای `get_queryset` یک مدیر `QuerySet` را اجرا می‌کند. ما این روش را رد می‌کنیم تا فیلتر سفارشی خود را در `QuerySet` نهایی وارد کنیم. ما مدیر دلخواه خود را تعریف کرده ایم و آن را به مدل `Post` اضافه کرده ایم؛ اکنون می‌توانیم از آن برای انجام سوالات استفاده کنیم. بیایید آن را تست کنیم.

مجدداً سورور توسعه را با دستور زیر راه انداری کنید:

```
python manage.py shell
```

اکنون می توانید تمام پستهای منتشر شده را که عنوان آنها با استفاده از دستور زیر شروع می شود ، بازیابی کنید:

```
Post.published.filter(title__startswith='Who')
```

Building list and detail views

اکنون که شما در مورد نحوه استفاده از ORM آگاهی دارید ، برای ساختن نظرات برنامه بلاگ آماده هستید. نمای Django فقط یک عملکرد CPython است که یک درخواست وب را دریافت می کند و یک پاسخ وب را برمی گرداند. تمام منطق بازگشت جواب دلخواه درون نمای قرار می گیرد.

ابتدا نمایهای برنامه خود را ایجاد می کنیم ، سپس برای هر نمای الگوی URL تعريف خواهیم کرد و در آخر ، قالب های HTML را برای ارائه داده های ایجاد شده توسط بازدیدها ایجاد خواهیم کرد. هر نمای یک الگوی متغیرهای منتقل شده به آن را ارائه می دهد و پاسخ HTTP را با خروجی ارائه شده برمی گرداند.

ایجاد لیست و نمایش جزئیات

بیایید با ایجاد یک نمایش برای نمایش لیست پست ها شروع کنیم. پرونده views.py برنامه وبلاگ خود را ویرایش کنید و به صورت زیر ظاهر شوید:

```
from django.shortcuts import render, get_object_or_404
from .models import Post

def post_list(request):
    posts = Post.published.all()
    return render(request,
                  'blog/post/list.html',
                  {'posts': posts})
```

شما تازه اولین نمای جنگو را ایجاد کرده اید. نمای post_list شیء درخواست را تنها پارامتر در نظر می گیرد. به یاد داشته باشید که این پارامتر توسط همه بازدیدها لازم است. در این دیدگاه ، ما با استفاده از مدیر منتشرشده ای که قبلاً ایجاد کرده بودیم ، همه پست ها را با وضعیت منتشر شده بازیابی می کنیم.

سرانجام ، ما از میانبر render ارائه شده توسط Django برای تهیه لیست پست ها با الگوی داده شده استفاده می کنیم. این تابع برای ارائه الگوی داده شده ، از هدف درخواست ، مسیر الگوی و متغیرهای متن استفاده می کند. این یک متن HttpResponse را با متن ارائه شده (به طور معمول کد HTML) بر می گرداند. میانبر render زمینه درخواست را در نظر می گیرد ، بنابراین هر متغیر تنظیم شده توسط پردازنده های متن قالب توسط الگوی داده شده قابل دسترسی است. پردازنده های متن قالب فقط فراخوانی می شوند که متغیرها را در متن قرار می دهند. شما در فصل 3 می اموزید که چگونه از این برای گسترش برنامه وبلاگ خود استفاده کنید.

باید برای نمایش یک پست نمای دوم ایجاد کنیم عملکرد زیر را به پرونده views.py اضافه کنید:

```
def post_detail(request, year, month, day, post):
    post = get_object_or_404(Post, slug=post,
                           status='published',
                           publish_year=year,
                           publish_month=month,
                           publish_day=day)
    return render(request,
                  'blog/post/detail.html',
                  {'post': post})
```

این نمای جزئیات پست است. این نمایش یک سال ، ماه ، روز و ارسال پارامترها طول می کشد تا بتوانید یک شی منتشر شده را با استفاده از تاریخ و زمانه مورد نظر خود بازگردد. توجه داشته باشید که وقتی مدل Post را ایجاد کردیم ، پارامتر slug را به قسمت theunique_for_date را اضافه کردیم. از این طریق اطمینان می دهیم که فقط یک پست برای یک تاریخ معین وجود خواهد داشت ، و بنابراین می توانیم پست های تک را با استفاده از تاریخ و Slug بازیابی کنیم. در نمای جزئیات ، ما از میانبر get_object_or_404 برای بازیابی پست مورد نظر استفاده می کنیم. این تابع شیء مطابق با پارامترهای داده شده را بازیابی می کند یا در صورت عدم یافتن جسم ، استثنائی HTTP 404 (یافت نشد) را راه اندازی می کند. در آخر ، ما از میانبر render() استفاده می کنیم تا پست بازیابی شده را با استفاده از یک الگوی ارائه دهیم.

اضافه کردن الگوهای URL برای بازدیدهای شما

الگوهای URL به شما امکان می دهد URL ها را برای مشاهده ها استفاده کنید. یک الگوی URL از یک الگوی رشته ای ، یک نمایش و ، اختیاری ، نامی تشکیل شده است که به شما امکان می دهد URL پروژه را به صورت گسترده بنویسید. جنگو از طریق هر الگوی URL اجرا می شود و در اولین مرحله که مطابق با URL درخواست شده است متوقف می شود. سپس ، جنگو نمای الگوی URL مطابق را وارد کرده و آن را اجرا می کند ، با نمونه ای از کلاس HttpRequest و کلمات کلیدی یا موقعیت های آرگومان. یک پرونده urls.py را در فهرست برنامه بلاگ ایجاد کنید و خطوط زیر را به آن اضافه کنید:

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

app_name = 'blog'

urlpatterns = [
    # post views
    path('', views.post_list, name='post_list'),
    path('<int:year>/<int:month>/<int:day>/<slug:post>/',
         views.post_detail,
         name='post_detail'),
]
```

در کد قبلی ، یک نام برنامه را با متغیر `app_name` تعریف می کنیم. این به ما امکان می دهد URL ها را به وسیله برنامه سازماندهی کرده و هنگام مراجعه به آنها از این نام استفاده کنیم. ما دو الگوی مختلف را با استفاده از عملکرد `(path)` تعریف می کنیم. الگوی اول URL هیچ استدلای ندارد و در نمای `post_list` نقشه برداری می شود. الگوی دوم چهار آرگومان زیر را می گیرد و در نمای `post_detail` نقشه برداری می شود:

سال: به یک عدد صحیح احتیاج دارد

ماه: به یک عدد صحیح احتیاج دارد

روز: به یک عدد صحیح احتیاج دارد

post: می تواند از کلمات و غیب ها تشکیل شود

ما از براکت های زاویه ای برای گرفتن مقادیر از URL استفاده می کنیم. هر مقدار مشخص شده در الگوی URL به عنوان `<parameter>` به عنوان یک رشته ضبط می شود. ما از مبدل های مسیر ، مانند `<int: year>` استفاده می کنیم تا به طور خاص یک عدد صحیح و `<slug: post>` را مطابقت داده و به آن برگردانیم تا بطور خاص با یک مثل حلزونی (یک رشته متشکل از حروف یا اعداد ASCII ، به علاوه کarakترهای underscore و hyphen) مطابقت داشته باشد. همه مبدل های مسیر ارائه شده توسط Django را می توانید در

<https://docs.djangoproject.com/en/2.0/topics/http/urls/#path-converters>

مشاهده کنید.

اگر استفاده از `path` و مبدل برای شما کافی نیست ، می توانید در عوض از `re_path` استفاده کنید تا الگوهای پیچیده URL را با عبارات منظم Python تعریف کنید. می توانید درباره تعریف الگوهای URL با اصطلاحات عادی در بیشتر بدانید

اید ، ممکن است ابتدا بخواهید نگاهی به بیان منظم HOWTO واقع در https://docs.djangoproject.com/en/2.0/ref/urls/#django.urls.re_path

<https://docs.python.org/3/howto/regex.html>

بیندازید.

حال باید الگوهای URL برنامه و بلاگ را در الگوهای اصلی URL پروژه درج کنید. پرونده urls.py را که در فهرست برنامه mysite پروژه شما قرار دارد ویرایش کنید و به صورت زیر ظاهر کنید:

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

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

الگوی URL جدید تعریف شده با شامل ، به الگوهای URL تعریف شده در برنامه و بلاگ اشاره دارد به طوری که آنها در زیر blog / path گنجانده شوند. ما این الگوهای را در و بلاگ فضای نام وارد می کنیم. فضاهای نام باید در کل پروژه شما بی نظیر باشد. بعداً ما با وارد کردن فضای نام ، به آدرس های اینترنتی و بلاگ خود به راحتی مراجعه خواهیم کرد ،

برای مثال ، ساختن blog:post_detail و blog:post_list URL مکانهای نام می توانید در مورد بدانید در <https://docs.djangoproject.com/fa/2.0/topics/http/urls/#url-namespaces>

URL های متعارف برای مدل ها

شما می توانید از URL post_detail که در قسمت قبل تعریف کرده اید استفاده کنید تا URL متعارف برای اشیاء Post ایجاد شود. کنوانسیون در جنگو اضافه کردن یک متاد get_absolute_url به مدلی است که آدرس متعارف شی را برمی گرداند. برای این روش از متند reverse() استفاده خواهیم کرد که به شما امکان می دهد URL ها را با نام آنها و عبور از پارامترهای اختیاری بسازید. پرونده yourmodels.py را ویرایش کنید و موارد زیر را اضافه کنید:

```
from django.urls import reverse

class Post(models.Model):
    # ...
    def get_absolute_url(self):
        return reverse('blog:post_detail',
                      args=[self.publish.year,
                            self.publish.month,
                            self.publish.day,
                            self.slug])
```

ما از پیوند `get_absolute_url` در قالب های خود برای پیوند دادن به پست های خاص استفاده خواهیم کرد.

ایجاد الگو برای دیدگاه های شما

ما برای برنامه وبلاگ بازدیدها و الگوهای URL ایجاد کرده ایم. اکنون زمان آن رسیده است که قالب هایی برای نمایش پست ها به روشنی کاربر پسند اضافه کنید. پوشه ها و پرونده های زیر را در فهرست برنامه وبلاگ خود ایجاد کنید:

```
templates/
  blog/
    base.html
    post/
      list.html
      detail.html
```

ساختار قبلی ساختار پرونده برای الگوهای ما خواهد بود.

فایل HTML ساختار اصلی وب سایت را شامل می شود و محتوا را به قسمت اصلی محتوا و نوار کناری تقسیم می کند. پرونده های details.html و list.html از پرونده base.html به ارث می برند تا لیست پست و بلاگ و مشخصات جزئیات به ترتیب ارائه شود. Django یک زیان قالب قدرتمند دارد که به شما امکان می دهد نحوه نمایش داده ها را مشخص کنید. این مبتنی بر برچسب های قالب ، متغیرهای قالب و فیلترهای قالب است:

برچسب های قالب ارائه عملکرد الگو را کنترل می کنند و شبیه { % tag % } هستند.

متغیرهای الگو هنگام ارائه الگو با مقادیر جایگزین می شوند و شبیه {{ variable }} نظر می رسد

فیلترهای قالب به شما امکان می دهند متغیرها را برای نمایش تغییر دهید و مانند {{}} filter

می توانید همه برچسب ها و فیلترهای قالب داخلی را در داخل مشاهده کنید

<https://docs.djangoproject.com/en/2.0/ref/templates/builtins/>.

بیایید فایل base.html را ویرایش کنیم و کد زیر را اضافه کنیم:

```
{% load static %}  
<!DOCTYPE html>  
<html>  
<head>  
    <title>{% block title %}{% endblock %}</title>  
    <link href="{% static "css/blog.css" %}" rel="stylesheet">  
</head>  
<body>  
    <div id="content">  
        {% block content %}  
        {% endblock %}  
    </div>  
    <div id="sidebar">  
        <h2>My blog</h2>  
        <p>This is my blog.</p>  
    </div>  
</body>  
</html>
```

می گوید که برچسب های قالب ثابت را که توسط برنامه پرونده های Django می گوید از فیلتر قالب `{% load static %}` به در تنظیمات INSTALLED_APPS ارائه شده است ، که در INSTALLED_APPS ارائه شده است بازگذاری کنید. پس از بارگیری آن ، می توانید از فیلتر قالب `{% static %}` در این الگو استفاده کنید. با استفاده از این فیلتر قالب می توانید پرونده های استاتیک مانند پرونده blog.css را در جای استاتیک / دایرکتوری برنامه و بلاگ مشاهده خواهید کرد. استاتیک / دایرکتوری را از کدی که همراه با این فصل است در همان محل پروژه خود کپی کنید تا بتوانید برگه های سبک CSS را اعمال کنید.

می توانید ببینید که دو برچسب `{% block %}` وجود دارد. اینها به جنگویی گویند که ما می خواهیم بلوک را در آن منطقه تعریف کنیم. قالب هایی که از این الگو به ارث می توانند بلوک ها را با محتوا پر کنند. ما یک بلوک به نام عنوان و یک بلوک به نام محتوا تعریف کرده ایم.

بیایید پرونده post / list.html را ویرایش کنیم و به صورت زیر ظاهر کنیم:

```
{% extends "blog/base.html" %}

{% block title %}My Blog{% endblock %}

{% block content %}
<h1>My Blog</h1>
{% for post in posts %}
<h2>
    <a href="{{ post.get_absolute_url }}">
        {{ post.title }}
    </a>
</h2>
<p class="date">
    Published {{ post.publish }} by {{ post.author }}
</p>
{{ post.body|truncatewords:30|linebreaks }}
{% endfor %}
{% endblock %}
```

با استفاده از برجسب قالب { extends theblog / base.html } ، به جنگوی گوی که از الگوی theblog / base.html به ارث برده شود. سپس ، ما عنوانین و بلوک های محتوای الگوی پایه را با محتوای موجود در پست ها پر می کنیم و عنوان ، تاریخ ، نویسنده و بدن آنها را نمایش می دهیم ، از جمله پیوندی در عنوان به URL متعارف پست. در بدن پست ، ما از دو فیلتر قالب استفاده می کنیم: کلمات کوتاهی مقدار را به تعداد کلمات مشخص شده تقسیم می کنند ، و linebreaks خط خروجی را به شکاف HTML تبدیل می کند. شما می توانید به همان اندازه که می خواهید فیلترهای قالب بندی کنید. هر یک به خروجی تولید شده توسط قبلی اعمال می شود. برای شروع سرور توسعه ، پوسته را باز کرده و دستور سرور management.py python را اجرا کنید. blog / Openhttp://127.0.0.1:8000 در مرورگر خود ، و همه موارد را مشاهده خواهید کرد. توجه داشته باشید که برای نشان دادن آنها در اینجا باید برخی از پست ها را با وضعیت Published داشته باشید. شما باید چیزی شبیه به این را ببینید:



My Blog

Who was Django Reinhardt?

Published Dec. 14, 2017, 8:54 a.m. by admin

Who was Django Reinhardt.

Another post

Published Dec. 14, 2017, 8:57 a.m. by admin

Post body.

My blog

This is my blog.

سپس ، اجازه دهید پرونده post / detail.html را ویرایش کنیم:

```
{% extends "blog/base.html" %}

{% block title %}{{ post.title }}{% endblock %}
```

```
{% block content %}
<h1>{{ post.title }}</h1>
<p class="date">
    Published {{ post.publish }} by {{ post.author }}
</p>
{{ post.body|linebreaks }}
{% endblock %}
```

اکنون می توانید به مرورگر خود برگردید و بر روی یک از عنوان های پست کلیک کنید تا نگاهی دقیق به یک پست بیندازید.
شما باید چیزی شبیه به این را ببینید:



به URL نگاهی بیندازید./blog/2017/12/14/who-was-djangoreinhardt/. ما URL های دوستانه SEO را برای پست های وبلاگ خود طراحی کرده ایم.

اضافه کردن صفحه بندی

با افزودن محتوا به وبلاگ خود ، به زودی متوجه خواهید شد که باید لیست پست ها را در چند صفحه تقسیم کنید. Django دارای یک کلاس طراحی داخلی است که به شما امکان می دهد داده های صفحه بندی شده را به راحتی مدیریت کنید.

برای وارد کردن طبقات Djangopaginator و اصلاح نمای views.py برنامه وبلاگ را به شرح زیر ویرایش کنید:

```

from django.core.paginator import Paginator, EmptyPage,\n                                         PageNotAnInteger\n\n\ndef post_list(request):\n    object_list = Post.published.all()\n    paginator = Paginator(object_list, 3) # 3 posts in each page\n    page = request.GET.get('page')\n    try:\n        posts = paginator.page(page)\n    except PageNotAnInteger:\n        # If page is not an integer deliver the first page\n        posts = paginator.page(1)\n    except EmptyPage:\n        # If page is out of range deliver last page of results\n        posts = paginator.page(paginator.num_pages)\n    return render(request,\n                  'blog/post/list.html',\n                  {'page': page,\n                   'posts': posts})

```

اینگونه کار می کند:

1. کلاس Paginator را با تعداد اشیایی که می خواهیم در هر صفحه نمایش دهیم ، قرار می دهیم.
 2. پارامتر صفحه GET را دریافت می کنیم که شماره صفحه فعلی را نشان می دهد.
 3. ما برای صفحه مورد نظر با استفاده از روش صفحه () Paginator تماس می کیریم.
 - 4- اگر پارامتر صفحه عدد صحیح نیست ، صفحه اول نتایج را بازیابی می کنیم. اگر این پارامتر عدد بالاتر از آخرین صفحه نتایج باشد ، صفحه آخر را بازیابی خواهیم کرد.
 - 5- شماره صفحه و اشیاء بازیابی شده را به قالب منتقل می کنیم.
- حال باید یک الگوی ایجاد کنیم تا صفحه اصلی را نمایش دهد تا در هر الگویی که از صفحه بندی استفاده می کند ، گنجانده شود. در قالب / پوشه برنامه وبلاگ ، یک پرونده جدید ایجاد کنید و نام آن را به زیان pagination.html pagination دهید. کد HTML زیر را به پرونده اضافه کنید:

```
<div class="pagination">
    <span class="step-links">
        {% if page.has_previous %}
            <a href="?page={{ page.previous_page_number }}>Previous</a>
        {% endif %}
        <span class="current">
            Page {{ page.number }} of {{ page.paginator.num_pages }}.
        </span>
        {% if page.has_next %}
            <a href="?page={{ page.next_page_number }}>Next</a>
        {% endif %}
    </span>
</div>
```

باید به الگوی بلاگ / post / list.html برگردیم و الگوی thepagination.html را در انتهای آن قرار دهیم

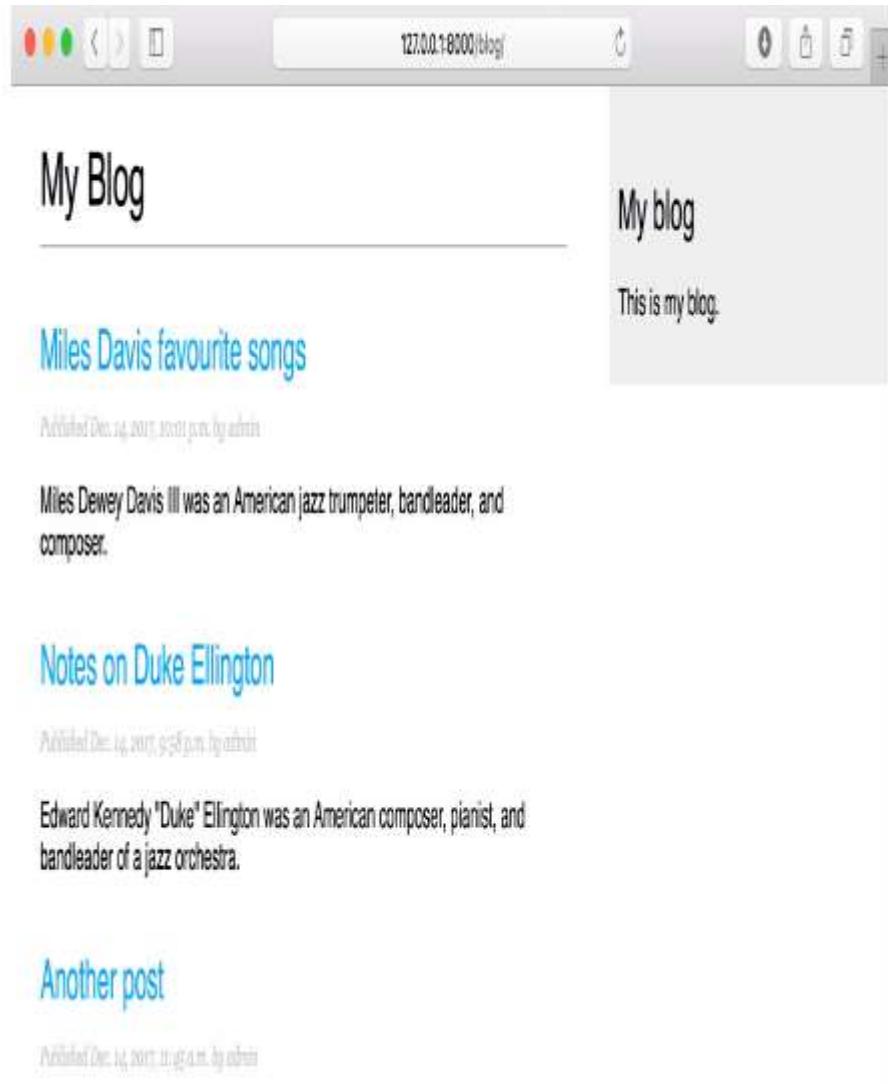
% content %} بلوک ، به شرح زیر است:

```
{% block content %}
...
{% include "pagination.html" with page=posts %}
{% endblock %}
```

از آنجا که به صفحه موردی که به آن الگوی رویم پست می گویند ، الگوی صفحه بندی را در الگوی لیست پست قرار می دهیم که پارامترها را عبور می دهد تا به درستی آن را ارائه دهد. برای استفاده مجدد از الگوریتم صفحه بندی خود در نماهای کوچک مدلها مختلف می توانید از این روش پیروی کنید.

اکنون در مرورگر خود <http://127.0.0.1:8000/blog/> را باز کنید.

باید صفحه بندی را در پایین لیست پست داشته باشد و بتوانید از طریق صفحات حرکت کنید:



استفاده از نماهای کلاس محور

نماهای مبتنی بر کلاس روشی جایگزین برای اجرای نماها به عنوان اشیاء پایتون به جای توابع است. از آنجاکه یک نمای ، تماس پذیر است که یک درخواست وب را می گیرد و پاسخ وب را برمی گرداند ، می توانید نظرات خود را به عنوان روش کلاس نیز تعریف کنید. جنگو کلاس های نمای پایه را برای این کار فراهم می کند. همه آنها از کلاس View به دست می آیند ، که دارای اعماق روش HTTP و سایر ویژگی های مشترک است.

نماهای مبتنی بر کلاس برای برخی از موارد استفاده مزایایی نسبت به نماهای مبتنی بر عملکرد دارد. آنها دارای ویژگی های زیر هستند:

سازماندهی کدهای مربوط به روش های HTTP مانند GET، POST با جای استفاده از انشعاب مشروط با استفاده از ارث های چندگانه برای ایجاد کلاس های نمای قابل استفاده مجدد (که به عنوان مخلوطین نیز شناخته می شوند) می توانید نگاهی به مقدمه کلاس بپردازید. بازدید در [./https://docs.djangoproject.com/fa/2.0/topics/class-based-views/intro](https://docs.djangoproject.com/fa/2.0/topics/class-based-views/intro)

برای استفاده از ListView عمومی ارائه شده توسط Django ، نمای post_list خود را به یک نمایش کلاس محور تغییر خواهیم داد. این نمای پایه به شما امکان می دهد اشیاء از هر نوع را لیست کنید.

پرونده views.py برنامه وبلاگ خود را ویرایش کرده و کد زیر را اضافه کنید:

```
from django.views.generic import ListView
```

```
class PostListView(ListView):
    queryset = Post.published.all()
    context_object_name = 'posts'
    paginate_by = 3
    template_name = 'blog/post/list.html'
```

این نمای کلاس مبتنی بر نمای post_list قبلی است.

در کد قبلی ، ما از لیست View می خواهیم موارد زیر را انجام دهد: به جای بازیابی همه اشیاء ، از QuerySet خاص استفاده کنید. به جای تعریف یک ویژگی queryset ، می توانستیم Post = model را مشخص کنیم و Django می توانست QuerySet عمومی all () را برای ما ایجاد کند. اگر از متن متغیر متن استفاده کنیم برای نتایج پرس و جو متغیر پیش فرض ، اگر شما هیچ متن object_name را مشخص نکنیم متغیر پیش فرض list object_list است. نتیجه را نمایش دهید که سه شی در هر صفحه نشان داده می شود. برای ارائه صفحه از یک الگوی سفارشی استفاده کنید. اگر یک الگوی پیش فرض تنظیم نکنیم ، ListView از بلاگ / post_list.html استفاده می کند.

اکنون پرونده urls.py برنامه و بلاگ خود را باز کنید ، در مورد الگوی URL پست_لیست قبلي اظهار نظر کنید و با استفاده از کلاس PostListView یک الگوی URL جدید اضافه کنید:

```
urlpatterns = [
    # post views
    # path('', views.post_list, name='post_list'),
    path('', views.PostListView.as_view(), name='post_list'),
    path('<int:year>/<int:month>/<int:day>/<slug:post>/',
        views.post_detail,
        name='post_detail'),
]
```

برای ادامه کار در صفحه بندی ، باید از شی صفحه مناسب استفاده کنیم که به قالب منتقل شده است. نمای کلی صفحه انتخاب شده را در متغیری به نام page_obj منتقل می کند ، بنابراین شما باید متناسب با آن الگوی پست / list.html خود را ویرایش کنید تا بتوانید paginator را با استفاده از متغیر مناسب به شرح زیر وارد کنید:

```
{% include "pagination.html" with page=page_obj %}
```

/ را در مرورگر خود باز کنید و تأیید کنید که همه چیز به همان روشه است که با post_listview قبلی کار می کند.

این یک نمونه ساده از یک دیدگاه مبتنی بر کلاس است که از یک کلاس عمومی تهیه شده توسط Django استفاده می کند. در بخش 10 ، ساختن یک بستر الکترونیکی و فصلهای پی در پی ، درباره دیدگاههای مبتنی بر کلاس اطلاعات بیشتری کسب خواهید کرد.

خلاصه

در این فصل ، شما با ایجاد یک برنامه وبلاگ اساسی ، اصول اولیه چارچوب وب Django را آموخته اید. شما مدل های داده ها را طراحی کرده اید و مهاجرت به پروژه خود را اعمال کرده اید. شما نمایش ها ، قالب ها و URL ها را برای وبلاگ خود ایجاد کرده اید ، از جمله صفحه بندی شی.

در فصل بعد ، شما یاد می گیرید که چگونه برنامه وبلاگ خود را با یک سیستم نظر و عملکرد برچسب گذاری تقویت کنید و به کاربران خود اجازه دهید پست ها را از طریق ایمیل به اشتراک بگذارند.

فصل دوم

ارتقاء وبلاگ

در فصل قبل ، شما یک برنامه اصلی وبلاگ ایجاد کرده اید. اکنون ، برنامه های خود را به یک وبلاگ کاملاً کاربردی با ویژگی های پیشرفته ، مانند اشتراک گذاری پست ها از طریق ایمیل ، اضافه کردن نظرات ، برچسب زدن به پست ها و بازیابی پست ها با شباهت تبدیل می کنید. در این فصل مباحث زیر را یاد خواهید گرفت:

ارسال ایمیل با Django

ایجاد اشکال و استفاده از آنها در نماها

ایجاد فرم از مدل ها

ادغام برنامه های شخص ثالث

ساخت QuerySets پیچیده

اشتراك گذاري پست ها از طریق ایمیل

ابتدا ، ما به کاربران این امکان را می دهیم که با ارسال ایمیل برای آنها پست ارسال کنند.

کم وقت بگذارید تا با استفاده از آنچه در فصل قبل آموخته اید ، از نحوه مشاهده ، URL و قالب برای ایجاد این قابلیت استفاده کنید ، اکنون فکر کنید ، برای اینکه به کاربران خود اجازه ارسال پیام از طریق ایمیل را بدهید ، آنچه را تیاز دارید بررسی کنید. برای انجام موارد زیر:

برای پر کردن نام و ایمیل خود ، دریافت کننده ایمیل و نظرات اختیاری ، یک فرم ایجاد کنید تا در پرونده views.py که داده های ارسال شده را به همراه دارد ، یک ایمیل ایجاد کنید و یک ایمیل را برای مشاهده جدید در urls.py ارسال کنید. پرونده برنامه وبلاگ برای نمایش فرم ایجاد شده است

ایجاد فرم با جنگو

بیایید با ساختن فرم برای به اشتراک گذاشتن پست ها شروع کنیم. Django چارچوبی از فرم داخلی دارد که به شما امکان می دهد فرم ها را به روشنی آسان ایجاد کنید.

چارچوب فرم به شما امکان می دهد زمینه های فرم خود را تعریف کنید ، نحوه نمایش آنها را مشخص کنید و نحوه تأیید اعتبار داده های ورودی را مشخص کنید. چارچوب فرم Django روشی انعطاف پذیر برای ارائه فرم ها و رسیدگی به داده ها ارائه می دهد.

جنگو با دو کلاس پایه برای ساختن فرم ها همراه است:

Form: به شما امکان می دهد فرم های استاندارد بسازید

ModelForm: به شما امکان می دهد فرم های گره خورده با نمونه های مدل بسازید

ابتدا ، یک فایل form.py را در فهرست پوشه برنامه و بلاگ خود ایجاد کنید

و آنرا مانند این شکل دهید:

```
from django import forms

class EmailPostForm(forms.Form):
    name = forms.CharField(max_length=25)
    email = forms.EmailField()
    to = forms.EmailField()
    comments = forms.CharField(required=False,
                                widget=forms.Textarea)
```

این اولین فرم جنگو است. به کد نگاهی بیندازید. ما با ارت بردن کلاس فرم ، یک فرم ایجاد کرده ایم. برای اعتبار سنجی زمینه ها ، از انواع زمینه های مختلف برای جنگو استفاده می کنیم.

قسمت نام CharField است. این نوع فیلد به عنوان <input type = "text"> عنصر HTML ارائه می شود. هر نوع فیلد دارای یک ویجت پیش فرض است که نحوه ارائه این زمینه در HTML را تعیین می کند. ویجت پیش فرض را می توان با ویژگی ویجت نادیده گرفت. در قسمت نظرات ، از یک ابزارک Textarea استفاده می کنیم تا به جای عنصر پیش فرض <input> ، آن را به عنوان <textarea> عنصر HTML نمایش دهیم.

اعتبار سنجی زمینه نیز به نوع فیلد بستگی دارد. به عنوان مثال، ایمیل و قسمتهای مربوط به فیلدهای EmailField است. هر دو قسمت به یک آدرس ایمیل معتبر احتیاج دارند، در غیر این صورت، اعتبار سنجی زمینه باعث ایجاد ValidationError استثناء شده و فرم اعتبار نخواهد یافت. پارامترهای دیگر نیز برای اعتبار سنجی فرم در نظر گرفته می‌شوند: ما حداقل طول 25 کاراکتر را برای قسمت نام تعریف می‌کنیم و قسمت نظرات را با required=False اختیاری می‌کنیم. همه این موارد برای اعتبار سنجی زمینه نیز در نظر گرفته شده است. انواع فیلد های مورد استفاده در این فرم تنها بخشی از زمینه های فرم جنگو است. برای لیستی از تمام زمینه های فرم موجود، می‌توانید به <https://docs.djangoproject.com/en/2.0/ref/forms/fields> مراجعه کنید.

رسیدگی به اشکال در نماها

شما باید یک نمای جدید ایجاد کنید که فرم را به دست آورد و هنگام ارسال با موفقیت ایمیل ارسال کنید. پرونده views.py بینامه وبلاگ خود را ویرایش کنید و کد زیر را به آن اضافه کنید:

این نمای به شرح زیر است:

ما نمای post_share را تعریف می کنیم که شیء درخواست و متغیر post_id را به عنوان پارامتر در نظر می گیرد.

برای بازیابی پست byID از میانبر get_object_or_404 () استفاده کنید و مطمئن شوید که پست بازیابی وضعیت منتشر شده دارد.

برای نمایش دادن فرم اولیه و پردازش داده های ارسالی ، از همان نمایش استفاده کنید. ما فرق می کنیم فرم ارائه شده است یا نه بر اساس روش درخواست و فرم را با استفاده از POST ارسال می کنیم. فرض می کنیم اگر یک درخواست GET دریافت کنیم ، باید یک فرم خالی نمایش داده شود و اگر درخواست POST دریافت کنیم ، فرم ارسال می شود و نیاز به پردازش دارد. بنابراین ، ما از درخواست 'POST' استفاده می کنیم. برای تمایز بین دو سناریو.

در زیر مراحل نمایش و رسیدگی به فرم آورده شده است:

1. هنگامی که نمایش در ابتدا با درخواست GET بارگیری می شود ، یک نمونه فرم جدید ایجاد می کنیم که برای نمایش فرم خالی در قالب استفاده می شود:

```
form = EmailPostForm()
```

2. کاربر فرم را پر کرده و آن را از طریق POST ارسال می کند. سپس با استفاده از داده های ارسالی که در درخواست موجود است یک نمونه فرم ایجاد می کنیم.

```
if request.method == 'POST':
    # Form was submitted
    form = EmailPostForm(request.POST)
```

3. پس از این ، داده های ارسال شده را با استفاده از روش() is_valid فرم تأیید می کنیم. این روش داده های معرفی شده در فرم را تأیید می کند و اگر همه قسمت ها دارای داده های معتبر باشند ، True را برمی گرداند. اگر هر فیلد حاوی داده های نامعتبر است ، آنگاه() is_valid غلط را برمی گرداند. با دسترسی به form.errors می توانید لیستی از خطاهای اعتبار سنجی را مشاهده کنید.

4- اگر فرم معتبر نیست ، مجدداً فرم را در قالب داده با داده های ارسال شده ارائه می دهیم. ما خطاهای اعتبار سنگی را در قالب الگو نمایش خواهیم داد.

5- اگر فرم معتبر است ، داده های معتبر را می توان به `accessingform.cleaned_data` بازیابی کرد. این ویژگی یک فرهنگ لغت از زمینه های فرم و مقادیر آنها است.

اگر داده های فرم شما اعتبار نداشته باشند ، `clean_data` فقط زمینه های معتبری را شامل می شود.
حال باید بیاموزیم که چگونه با استفاده از جنگو ایمیل بفرستیم تا همه چیز در کنار هم قرار بگیرد.

ارسال ایمیل با Django

ارسال ایمیل با Django بسیار ساده است. ابتدا باید یک سرور SMTP محلی داشته باشید یا پیکربندی یک سرور SMTP خارجی را با اضافه کردن تنظیمات زیر در فایل `startsettings.py` پروژه خود تعریف کنید:

EMAIL_HOST: میزبان سرور SMTP؛ پیش فرض محلی است

EMAIL_PORT: درگاه SMTP؛ پیش فرض 25 است

EMAIL_HOST_USER: نام کاربری سرور SMTP

EMAIL_HOST_PASSWORD: رمز عبور برای سرور SMTP

EMAIL_USE_TLS: استفاده از یک اتصال امن TLS

EMAIL_USE_SSL: استفاده از یک اتصال ایمن TLS ضمیمی

اگر نمی توانید از سرور SMTP استفاده کنید ، می توانید با افزودن تنظیمات زیر به `setting.py` بنویسید:

```
EMAIL_BACKEND = 'django.core.mail.backends.console.EmailBackend'
```

با استفاده از این تنظیمات ، جنگو تمام ایمیل های موجود در پوسته را صادر می کند. این برای تست برنامه شما بدون سرور SMTP بسیار مفید است.

اگر می خواهید ایمیل بفرستید ، اما سرور محلی SMTP ندارید ، احتمالاً می توانید از سرور SMTP ارائه دهنده خدمات ایمیل خود استفاده کنید. پیکربندی نمونه زیر برای ارسال ایمیل از طریق سرورهای Gmail با استفاده از یک حساب Google معتبر است:

```
EMAIL_HOST = 'smtp.gmail.com'  
EMAIL_HOST_USER = 'your_account@gmail.com'  
EMAIL_HOST_PASSWORD = 'your_password'  
EMAIL_PORT = 587  
EMAIL_USE_TLS = True
```

برای باز کردن پوسته پایتون و ارسال ایمیل به شرح زیر ، دستور shell python manage.py را اجرا کنید:

```
>>> from django.core.mail import send_mail  
>>> send_mail('Django mail', 'This e-mail was sent with Django.',  
'your_account@gmail.com', ['your_account@gmail.com'], fail_silently=False)
```

تابع () send_mail موضوع ، پیام ، فرستنده و لیست گیرنده آرگومان های مورد نیاز در نظر می گیرد. با تنظیم آرگومان اختیاری fail_silently=False ، به ما می گوید اگر ایمیل نتواند به درستی ارسال شود ، استثنائی را ایجاد می کند. اگر خروجی که می بینید 1 است ، ایمیل شما با موفقیت ارسال شد.

اگر با پیکربندی قبلی ایمیل های خود را توسط Gmail ارسال می کنید ، ممکن است مجبور شوید دسترسی به برنامه های با امنیت کمتر را در <https://myaccount.google.com/lesssecureapps> به شرح زیر انجام دهید:

Some apps and devices use less secure sign-in technology, which makes your account more vulnerable. You can turn off access for these apps, which we recommend, or turn on access if you want to use them despite the risks. [Learn more](#)

Allow less secure apps: ON



اکنون ، ما این عملکرد را به نظر خود اضافه خواهیم کرد.

نمای post_share را در پرونده views.py برنامه و بلاگ به شرح زیر ویرایش کنید:

```
from django.core.mail import send_mail

def post_share(request, post_id):
    # Retrieve post by id
    post = get_object_or_404(Post, id=post_id, status='published')
    sent = False

    if request.method == 'POST':
        # Form was submitted
        form = EmailPostForm(request.POST)
        if form.is_valid():
            # Form fields passed validation
            cd = form.cleaned_data
            post_url = request.build_absolute_uri(
                post.get_absolute_url())
            subject = '{0} ({1}) recommends you reading "'
            subject += '{2}'.format(cd['name'], cd['email'], post.title)
            message = 'Read "{0}" at {1}\n\n{2}\''s comments:'
            message += '{3}'.format(post.title, post_url, cd['name'], cd['comments'])
            send_mail(subject, message, 'admin@myblog.com',
                      [cd['to']])
            sent = True
    else:
        form = EmailPostForm()
    return render(request, 'blog/post/share.html', {'post': post,
                                                    'form': form,
                                                    'sent': sent})
```

ما متغیر ارسال شده را اعلام می کنیم و هنگام ارسال پست آن را روی True قرار می دهیم.

بعداً در قالب الگو از آن متغیر استفاده می کند تا فرم موفقیت آمیز شود. از آنجا که ما باید یک لینک به پست را در ایمیل وارد کنیم ، ما مسیر مطلق ارسال را با استفاده از روش() get_absolute_url آن بازیابی خواهیم کرد. از این مسیر به عنوان ورودی برای درخواست استفاده کنید . () build_absolute_uri برای ایجاد یک URL کامل ، از جمله برنامه HTTP و نام میزبان. ما موضوع و بدنہ پیام ایمیل را با استفاده از داده های تمیز شده از فرم معتبر می سازیم و در آخر ایمیل را به آدرس ایمیل موجود در قسمت مربوط به فرم ارسال می کنیم.

اکنون که نمای شما کامل است ، به یاد داشته باشید که الگوی URL جدیدی را برای آن اضافه کنید. در پرونده urls.py برنامه و بلاگ خود را باز کرده و الگوی URL post_share را به شرح زیر اضافه کنید:

```
urlpatterns = [
    # ...
    path('<int:post_id>/share/',
         views.post_share, name='post_share'),
]
```

ارائه فرم در قالب

پس از ایجاد فرم ، برنامه نویسی نمای و اضافه کردن الگوی URL ، ما فقط قالب این نمایش را از دست نمی دهیم. یک فایل جدید در share.html ایجاد کرده و blog/templates/blog/post/ directory را نامگذاری کنید. کد زیر را به آن اضافه کنید:

```
{% extends "blog/base.html" %}

{% block title %}Share a post{% endblock %}

{% block content %}
  {% if sent %}
    <h1>E-mail successfully sent</h1>
    <p>
      "{{ post.title }}" was successfully sent to {{ form.cleaned_data.to }}.
    </p>
  {% else %}
    <h1>Share "{{ post.title }}" by e-mail</h1>
    <form action"." method="post">
      {{ form.as_p }}
      {% csrf_token %}
      <input type="submit" value="Send e-mail">
    </form>
  {% endif %}
{% endblock %}
```

این الگویی برای نمایش فرم یا پیام موفقیت آمیز هنگام ارسال است. همانطور که ملاحظه می کنید ، ما عنصر فرم HTML را ایجاد می کنیم ، نشان می دهد که باید با روش POST ارسال شود:

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

سپس نمونه واقعی فرم را به Django می گوییم که زمینه های خود را در عناصر پاراگراف <p> با روش as_p ارائه دهد. همچنین می توانیم فرم را به عنوان یک لیست بدون هماهنگی با as_ul یا یک جدول HTML با as_table ارائه دهیم. اگر می خواهیم هر زمینه ای را ارائه دهیم ، می توانیم

```
{% for field in form %}  
  <div>  
    {{ field.errors }}  
    {{ field.label_tag }} {{ field }}  
  </div>  
{% endfor %}
```

برچسب قالب { % csrf_token % } یک زمینه پنهان را با یک نشانه خودکار تولید می کند تا از حملات درخواست جعلی (CSRF) در سطح سایت جلوگیری کند. این حملات شامل یک وب سایت یا برنامه مخرب است که یک عمل ناخواسته را برای کاربر در سایت شما انجام می دهد.

می توانید اطلاعات بیشتر در مورد این را در

می توانید اطلاعات بیشتر در مورد این را در [https://www.owasp.org/index.php/Cross-site_Request_Forgery_\(CSRF\)](https://www.owasp.org/index.php/Cross-site_Request_Forgery_(CSRF)) پیدا کنید.

برچسب قبلی یک فیلد پنهان ایجاد می کند که به نظر می رسد:

```
<input type='hidden' name='csrfmiddlewaretoken'  
value='26JjKo2lcEtYkGoV9z4XmJIEHLXN5LDR' />
```

الگوی blog/post/detail.html خود را ویرایش کنید و بعد از متغیر {{post.body | linebreaks}} پیوند زیر را به آدرس URL پست اشتراک اضافه کنید:

```
<p>  
  <a href="{% url "blog:post_share" post.id %}">  
    Share this post  
  </a>  
</p>
```

به یاد داشته باشید که ما URL را بطور پویا با استفاده از {{ }} می سازیم برچسب قالب ارائه شده توسط Django. ما از فضای نامی به نام وبلاگ و URL به نام post_share استفاده می کنیم و شناسه پست را به عنوان پارامتر برای ساختن URL مطلق در حال عبور هستیم.

اکنون ، سرور توسعه را با python manage.py شروع کنید

دستور runserver را بزنید و <http://127.0.0.1:8000/blog> را در مرورگر خود باز کنید.

برای دیدن صفحه جزئیات آن ، روی هر عنوان پست کلیک کنید. در قسمت زیر ، باید پیوندی را که اضافه کردیم ، همانطور که در تصویر زیر نشان داده شده است مشاهده کنید:

Notes on Duke Ellington

Published Dec. 14, 2017, 9:58 p.m. by admin

Edward Kennedy "Duke" Ellington was an American composer, pianist, and bandleader of a jazz orchestra.

[Share this post](#)

My blog

This is my blog.

بر روی اشتراک این پست کلیک کنید ، و باید صفحه مربوط به فرم را برای اشتراک گذاری این پست از طریق ایمیل مشاهده کنید ، به شرح زیر:

Share "Notes on Duke Ellington" by e-mail

Name:

Email:

To:

Comments:

SEND E-MAIL

My blog

This is my blog.

سبک های CSS برای فرم در کد مثال در پرونده static / css / blog.css گنجانده شده است. هنگامی که روی دکمه SEND E-MAIL کلیک می کنید ، فرم ارسال و اعتبار سنجی می شود. اگر همه قسمت ها حاوی داده های معتبر باشند ، یک پیام موفقیت آمیز به شرح زیر دریافت می کنید:

E-mail successfully sent

"Notes on Duke Ellington" was successfully sent to account@gmail.com.

My blog

This is my blog.

If you input invalid data, you will see that the form is rendered again, including all validation errors:

Share "Notes on Duke Ellington" by e-mail

Name:

Anonic

My blog

This is my blog.

* Enter a valid email address.

Email:

invalid

* This field is required.

To:

Comments:

SEND E-MAIL

توجه داشته باشید که برخی مروگرهای مدرن از ارسال فرم با قسمتهای خالی یا اشتباہ جلوگیری می کنند. این امر به دلیل اعتبارسنجی فرم است که توسط مروگر مبتنی بر انواع فیلدها و محدودیت ها در هر زمینه انجام می شود. در این حالت ، فرم ارسال نمی شود و مروگر پیغام خطای را برای قسمت های اشتباہ نشان می دهد.

فرم ما برای به اشتراک گذاشتن پست ها از طریق ایمیل اکنون کامل است. بیایید یک سیستم نظر برای وبلاگ خود ایجاد کنیم.

ایجاد سیستم کامنت

اکنون ، ما یک سیستم نظر برای وبلاگ ایجاد خواهیم کرد ، که در آن کاربران قادر به اظهار نظر در مورد پست ها خواهند بود. برای ساختن سیستم نظر ، باید مراحل زیر را انجام دهیم:

1. برای ذخیره نظرات ، مدلی را ایجاد کنید
2. برای ارسال نظر و اعتبار داده های ورودی ، فرم ایجاد کنید
3. نمایی را اضافه کنید که فرم را پردازش کرده و نظر جدید را به پایگاه داده ذخیره می کند
4. قالب جزئیات ارسال را ویرایش کنید تا لیستی از نظرات و فرم برای نمایش یک نظر جدید نمایش داده شود ، ابتدا اجازه دهیم یک مدل برای ذخیره نظرات بسازیم. پرونده model.py برنامه وبلاگ خود را باز کرده و کد زیر را اضافه کنید:

```

class Comment(models.Model):
    post = models.ForeignKey(Post,
                            on_delete=models.CASCADE,
                            related_name='comments')
    name = models.CharField(max_length=80)
    email = models.EmailField()
    body = models.TextField()
    created = models.DateTimeField(auto_now_add=True)
    updated = models.DateTimeField(auto_now=True)
    active = models.BooleanField(default=True)

    class Meta:
        ordering = ('created',)

    def __str__(self):
        return 'Comment by {} on {}'.format(self.name, self.post)

```

این مدل نظر ماست این شامل ForeignKey است که نظر را با یک پست واحد مرتبط می کند. این رابطه چند به یک در مدل نظر تعریف شده است زیرا هر نظر در یک پست انجام می شود و هر پست ممکن است چندین اظهار نظر داشته باشد. صفت Therelated_name به ما امکان می دهد صفتی را که برای رابطه از شیء مربوط به آن استفاده می کنیم ، به این ویژگی اختصاص دهیم. پس از تعریف این موضوع ، می توانیم پست یک شیء نظر را با استفاده از comcom.post کنیم و تمام نظرات یک پست را با استفاده از all () بدست آوریم. اگر ویژگی related_name را تعریف نکنید ، Django از نام مدل با حروف کوچک و به دنبال _set y (یعنی coment_set) استفاده می کند تا مدیر شیء مربوط را به این قسمت معرفی کند.

می توانید در مورد روابط بسیاری به یک اطلاعات بیشتری کسب کنید

[./https://docs.djangoproject.com/fa/2.0/topics/db/examples/many_to_one](https://docs.djangoproject.com/fa/2.0/topics/db/examples/many_to_one)

ما یک فیلد بولی فعال را گنجانده ایم که برای غیرفعال کردن دستی نظرات نامناسب از آن استفاده خواهیم کرد. ما از قسمت ایجاد شده برای مرتب سازی نظرات به صورت ترتیب به صورت پیش فرض استفاده می کنیم.

مدل جدید کامنت که شما فقط ایجاد کرده اید هنوز در بانک اطلاعاتی همگام نشده است. دستور زیر را برای ایجاد یک مهاجرت جدید که نشان دهنده ایجاد مدل جدید است اجرا کنید:

```
python manage.py makemigrations blog
```

باید خروجی زیر را مشاهده کنید:

```
Migrations for 'blog':  
  blog/migrations/0002_comment.py  
    - Create model Comment
```

جنگو یک پرونده comment.py_0002 را در داخل مهاجرت ها ایجاد کرده است

فهرست برنامه و بلاگ حالا باید برنامه پایگاه داده مربوط را ایجاد کرده و تعییرات را در دیتابیس اعمال کنید.

دستور زیر را برای اعمال مهاجرت های موجود اجرا کنید:

```
python manage.py migrate
```

شما خروجی دریافت خواهید کرد که شامل خط زیر است:

```
Applying blog.0002_comment... OK
```

مهاجرتی که اخیراً ایجاد کردیم اعمال شده است و اکنون جدول ablog_comment در بانک اطلاعاتی وجود دارد.

اکنون می توانیم مدل جدید خود را به سایت مدیریت اضافه کنیم تا بتوانیم از طریق یک رابط کاربری ساده نظرات را مدیریت کنیم. پرونده admin.py برنامه و بلاگ را باز کنید ، مدل نظر را وارد کنید و کلاس ModelAdmin زیر را اضافه کنید:

```

from .models import Post, Comment

@admin.register(Comment)
class CommentAdmin(admin.ModelAdmin):
    list_display = ('name', 'email', 'post', 'created', 'active')
    list_filter = ('active', 'created', 'updated')
    search_fields = ('name', 'email', 'body')

```

سرور توسعه را با اجرای برنامه Python management.py شروع کنید

/http://127.0.0.1:8000/admin را در مرورگر خود باز کرده. همانطور که در تصویر زیر نشان داده شده است ، باید مدل جدید موجود در بخش BLOG را مشاهده کنید:

Comments	Add	Change

Posts	Add	Change

هم اکنون این مدل در سایت مدیریت ثبت شده است و ممکن است توانیم آنرا مدیریت کنیم موارد را با استفاده از یک رابط ساده کامنت کنید.

ایجاد فرم از مدل ها

ما هنوز نیاز به ایجاد یک فرم داریم تا کاربران بتوانیم در مورد پست های وبلاگ اظهار نظر کنند. به یاد داشته باشید که جنگو دو کلاس پایه برای ساختن فرم ها ، فرم و ModelForm دارد. شما از اولین مورد قبلی استفاده کرده اید تا به کاربران خود اجازه دهید پست ها را از طریق ایمیل به اشتراک بگذارند. در این مورد ، شما نیاز به استفاده از ModelForm دارید زیرا باید یک فرم را بصورت پویا از مدل نظر خود بسازید. فایل form.py برنامه وبلاگ خود را ویرایش کنید و خطوط زیر را اضافه کنید:

```
from .models import Comment

class CommentForm(forms.ModelForm):
    class Meta:
        model = Comment
        fields = ('name', 'email', 'body')
```

برای ایجاد یک فرم از یک مدل ، فقط لازم است که نشان دهیم از کدام مدل برای ساخت فرم در کلاس متأ فرم استفاده کنیم. جنگو مدل را درون تصویر می کند و فرم را به صورت پویا برای ما می سازد. هر نوع فیلد مدل دارای نوع فیلد فرم پیش فرض مربوطه است.

نحوه تعریف زمینه های مدل ما برای اعتبار سنجی فرم در نظر گرفته شده است. به طور پیش فرض ، جنگو برای هر فیلد موجود در مدل یک فیلد فرم ایجاد می کند. با این وجود ، شما می توانید با استفاده از لیست فیلدها ، چارچوبهایی را که می خواهید در فرم خود درج کنید ، بطور صریح بیان کنید یا اینکه با استفاده از لیست فیلدهای حذف شده ، کدام قسمتها را حذف کنید. برای فرم نظر ما ، فقط از نام ، ایمیل و فیلدهای body استفاده خواهیم کرد زیرا این تنها فیلدهایی هستند که کاربران ما قادر به پر کردن آن خواهند بود.

رسیدگی به ModelForms در نماها

ما از نمای جزئیات پست برای فرم و پردازش آن به منظور ساده نگه داشتن استفاده خواهیم کرد. پرونده views.py را ویرایش کنید، واردات را برای مدل نظر و فرم نظر اضافه کنید و نمای post_detail را اصلاح کنید تا به صورت زیر ظاهر شود:

```
from .models import Post, Comment
from .forms import EmailPostForm, CommentForm

def post_detail(request, year, month, day, post):
    post = get_object_or_404(Post, slug=post,
                           status='published',
                           publish__year=year,
                           publish__month=month,
                           publish__day=day)

    # List of active comments for this post
    comments = post.comments.filter(active=True)

    new_comment = None

    if request.method == 'POST':
        # A comment was posted
        comment_form = CommentForm(data=request.POST)
        if comment_form.is_valid():
            # Create Comment object but don't save to database yet
            new_comment = comment_form.save(commit=False)
            # Assign the current post to the comment
            new_comment.post = post
            # Save the comment to the database
            new_comment.save()
    else:
        comment_form = CommentForm()

    return render(request,
                  'blog/post/detail.html',
                  {'post': post,
                   'comments': comments,
                   'new_comment': new_comment,
                   'comment_form': comment_form})
```

بیایید آنچه را که به دیدگاه ما اضافه شده است مرور کنیم. ما از نمای post_detail برای نمایش پست و نظرات آن استفاده کردیم. ما یک QuerySet اضافه کردیم تا همه نظرات فعال را برای این پست بازیابی کنیم، به شرح زیر:

```
comments = post.comments.filter(active=True)
```

ما این QuerySet را با شروع از شیء پست می سازیم. از مدیر برای اشیاء مرتبط استفاده کنید که ما با استفاده از ویژگی related_name رابطه در مدل دیدگاه تعریف کردیم.

ما همچنین از همین نظر استفاده می کنیم تا به کاربران خود اجازه دهیم نظر جدیدی اضافه کنند. بنابراین ، متغیر new_comment را با تنظیم کردن آن در یک مقدار اولیه ، اولیه می کنیم. وقتی یک نظر جدید ایجاد شد ، از این متغیر استفاده خواهیم کرد. اگر نمایی با درخواست GET فراخوانی شود ، ما یک نمونه فرم را با comment_form = ایجاد می کنیم. اگر درخواست از طریق POST انجام شده است ، ما فرم را با استفاده از داده های CommentForm ارسالی فوری می کنیم و آن را با استفاده از روش is_valid اعتبار می دهیم.

اگر فرم نامعتبر است ، ما با خطاهای اعتبارسنجی الگورایمی دهیم. اگر فرم معتبر است ، اقدامات زیر را انجام می دهیم:

1. با فراخوانی () فرم، فرم مورد نظر جدید ایجاد می کنیم.

روش و اختصاص آن به متغیر new_comment به شرح زیر است:

```
new_comment = comment_form.save(commit=False)
```

روش save () نمونه ای از مدل را ایجاد می کند که فرم به آن پیوند داده شده و آن را در بانک اطلاعات ذخیره می کند. اگر آن را با استفاده از خطای False می خوانید ، نمونه مدل را ایجاد می کنید ، اما هنوز آن را در بانک اطلاعات ذخیره نکردید. این کار در شرایطی مفید خواهد بود که می خواهید قبل از ذخیره در آن ، آن چیزی را انجام دهید که در مرحله بعد آن را ذخیره کنید.

2. ما پست فعلی را به نظری که تازه ایجاد کردیم اختصاص می دهیم:

```
new_comment.post = post
```

با این کار مشخص می کنیم که نظر جدید متعلق به این پست است.

3. در آخر با استفاده از روش save () آن ، نظر جدید را در پایگاه داده ذخیره می کنیم:

```
new_comment.save()
```

اکنون دیدگاه ما آماده نمایش و پردازش نظرات جدید است.

افزودن نظرات به الگوی جزئیات ارسال

ما عملکردی را برای مدیریت نظرات برای یک پست ایجاد کرده ایم.

حال برای انجام موارد زیر باید به الگوی post / details.html خودمان را تطبیق دهیم:

نمایش تعداد کل نظرات برای پست

نمایش لیست نظرات

برای افزودن یک نظر جدید ، یک فرم برای کاربران نمایش دهید

ابتدا کامنت ها را اضافه خواهیم کرد. post / detail.htmltemplate را باز کنید و کد زیر را در بخش محتوای آن اضافه کنید:

```
{% with comments.count as total_comments %}  
  <h2>  
    {{ total_comments }} comment{{ total_comments|pluralize }}  
  </h2>  
{% endwith %}
```

ما در قالب الگوی از Django ORM استفاده می کنیم. توجه داشته باشید که زبان الگوی جنگو برای روش های فراخوانی از پرانتز استفاده نمی کند. برچسب {٪ with } به ما این امکان را می دهد تا مقداری را به متغیر جدید اختصاص دهیم تا در صورت استفاده از برچسب {٪ endwith } قابل استفاده باشد.

ما از فیلتر قالب جمع استفاده می کنیم بسته به مقدار total_comments ، یک پسوند جمع برای کلمه نظر نمایش می دهیم. فیلترهای قالب مقدار متغیری را که به عنوان ورودی به آنها اعمال می شود ، می گیرند و یک مقدار محاسبه شده را برای گردانند. ما در فصل 3 ، گسترش برنامه وبلاگ شما ، در مورد فیلترهای قالب بحث خواهیم کرد.

فیلتر قالب جمع ، رشتہ ای را با حرف "s" در صورت تغییر مقدار از 1 برای گرداند. متن قبل به صورت 0 کامنت ، 1 نظر یا N نظر ارائه می شود. جنگو شامل برچسب ها و فیلترهای قالب زیادی است که به شما کمک می کنند اطلاعات را به رویی که می خواهید نمایش دهید.

حال ، اجازه دهید لیست نظرات را درج کنیم. خطوط زیر را به الگوی post / detail.html در زیر کد قبلی اضافه کنید:

```
{% for comment in comments %}
<div class="comment">
    <p class="info">
        Comment {{ forloop.counter }} by {{ comment.name }}
        {{ comment.created }}
    </p>
    {{ comment.body|linebreaks }}
</div>
{% empty %}
    <p>There are no comments yet.</p>
{% endfor %}
```

ما برای برطرف کردن نظرات از برچسب قالب {٪ for } استفاده می کنیم. اگر لیست نظرات خالی باشد ، یک پیام پیش فرض را نمایش می دهیم ، و به کاربرانمان اطلاع می دهیم که هنوز هیچ نظری درباره این پست وجود ندارد. ما با متغیر {{forloop.counter}} ، که شامل شمارنده حلقه در هر تکرار است ، شمارش می کنیم. سپس نام کاربری که نظر را ارسال کرده ، تاریخ و بدنہ نظر را نمایش می دهیم.

سرانجام ، هنگام ارسال موفقیت آمیز ، باید فرم را ارائه داده یا پیام موفقی را به نمایش بگذارید خطوط زیر را درست زیر کد قبلی اضافه کنید:

```
{% if new_comment %}  
    <h2>Your comment has been added.</h2>
```

```
{% else %}  
    <h2>Add a new comment</h2>  
    <form action="." method="post">  
        {{ comment_form.as_p }}  
        {% csrf_token %}  
        <p><input type="submit" value="Add comment"></p>  
    </form>  
{% endif %}
```

کد بسیار ساده است: اگر جسم new_comment وجود داشته باشد ، ما یک پیام موفقیت آمیز نشان می دهیم زیرا این نظر با موفقیت ایجاد شد. در غیر این صورت ، فرم را با یک عنصر پاراگراف <p> برای هر فیلد ارائه می دهیم و نشان CSRF را برای درخواست های POST درج می کنیم. در مرورگر خود <http://127.0.0.1:8000/blog/> را در مرورگر خود باز کرده و روی عنوان عنوان کلیک کنید تا به صفحه جزئیات آن بروید. چیزی شبیه به تصویر زیر خواهد دید:

Notes on Duke Ellington

My blog

This is my blog.

Published 3 years ago by [eddie](#)
Edward Kennedy 'Duke' Ellington was an American composer, pianist, and bandleader of a jazz orchestra.

[Share this post!](#)

0 comments

There are no comments yet.

Add a new comment

Name:

Email:

Body:

[ADD COMMENT](#)

با استفاده از فرم ، چند نظر اضافه کنید. آنها باید ظاهر شوند

تحت پست خود را به ترتیب زمانی ، به شرح زیر:

2 comments

Comment 1 by Antonio Dec. 14, 2017, 10:08 p.m.

It's very interesting.

Comment 2 by Bienvenida Dec. 14, 2017, 10:09 p.m.

I didn't know that.

هایی که ایجاد کرده اید خواهید دید. برای ویرایش آن روی یک از آنها کلیک کنید ، کادر انتخاب فعال را بردارید و بر روی دکمه ذخیره کلیک کنید. شما دوباره به لیست نظرات هدایت می شوید و ستون Active یک نماد غیرفعال برای نظر نمایش می دهد.
باید مانند اولین نظر در تصویر زیر باشد:

Select comment to change

NAME	EMAIL	POST	CREATED	ACTIVE
Antonio	user1@gmail.com	Notes on Duke Ellington	Aug. 25, 2017, 5:08 p.m.	
Bienvenida	user2@gmail.com	Notes on Duke Ellington	Aug. 25, 2017, 5:08 p.m.	

2 comments

اگر به نمای جزئیات پست برگردید ، توجه خواهید کرد که نظر حذف شده دیگر نمایش داده نمی شود. همچنین برای تعداد کل نظرات شمارش نشده است. با تشکر از زمینه فعال ، می توانید نظرات نامناسب را غیرفعال کنید و از نمایش آنها در پست های خودداری کنید.

اضافه کردن قابلیت برچسب زدن

پس از اجرای سیستم اظهار نظر ، راهی برای برچسب زدن به پست های ما ایجاد خواهید کرد. شما این کار را با یکپارچه سازی یک برنامه برچسب گذاری جنگل شخص ثالث در پروژه ما انجام خواهید داد. ماژول دارای برچسب Django یک

برنامه قابل استفاده مجدد است که در ابتدای یک مدل Tag و یک مدیر را برای شما فراهم می کند تا برچسب ها را به راحتی به هر مدل اضافه کنید. می توانید به کد منبع آن در <https://github.com/alex/django-taggit> نگاهی بیندازید.

ابتدا باید با اجرای دستور زیر ، برچسب های جنگو را از طریق پیپ نصب کنید:

```
pip install django_taggit==0.22.2
```

سپس پرونده setting.py را باز کرده و برچسب git را به تنظیمات INSTALLED_APPS خود اضافه کنید ،
به شرح زیر:

```
INSTALLED_APPS = [
    # ...
    'blog.apps.BlogConfig',
    'taggit',
]
```

پرونده model.py برنامه و بلاگ خود را باز کنید و مدیر TaggableManager تهیه شده توسط Django-taggit را با استفاده از کد زیر به مدل Post اضافه کنید:

```
from taggit.managers import TaggableManager

class Post(models.Model):
    # ...
    tags = TaggableManager()
```

مدیر برچسب ها به شما امکان اضافه کردن ، بازیابی و حذف برچسب ها از اشیاء Post را می دهد. دستور زیر را اجرا کنید تا یک تغییر برای تغییر مدل خود ایجاد کنید:

```
python manage.py makemigrations blog
```

شما باید خروجی زیر را بدست آورید:

```
Migrations for 'blog':  
  blog/migrations/0003_post_tags.py  
    - Add field tags to post
```

اکنون ، دستور زیر را برای ایجاد جداول بانک اطلاعاتی مورد نیاز برای مدل های Django-taggit و همگام سازی تغییرات مدل خود اجرا کنید:

```
python manage.py migrate
```

خروجی را مشاهده خواهید کرد که نشان می دهد مهاجرت به شرح زیر اعمال شده است:

```
Applying taggit.0001_initial... OK  
Applying taggit.0002_auto_20150616_2121... OK  
Applying blog.0003_post_tags... OK
```

بانک اطلاعاتی شما اکنون آماده استفاده از مدل های Django-taggit است. باید یاد بگیریم که چگونه از مدیر تگ ها استفاده کنیم. با دستور python manage.py پوسته را باز کرده و کد زیر را وارد کنید. اول ، ما یکی از پست های خود را (همان شماره 1 شناسه) بازیابی می کنیم:

```
>>> from blog.models import Post  
>>> post = Post.objects.get(id=1)
```

سپس ، برخی از برجسب ها را به آن اضافه کنید و برجسب های آن را بازیابی کنید تا بررسی کنید که آیا آنها با موفقیت اضافه شدند:

```
>>> post.tags.add('music', 'jazz', 'django')
>>> post.tags.all()
<QuerySet [<Tag: jazz>, <Tag: music>, <Tag: django>]>
```

در آخر ، یک برجسب را بردارید و دوباره لیست برجسب ها را بررسی کنید:

```
>>> post.tags.remove('django')
>>> post.tags.all()
<QuerySet [<Tag: jazz>, <Tag: music>]>
```

این آسان بود ، درست است؟ سرور python management.py را اجرا کنید دوباره سرور توسعه را شروع کنید و http://127.0.0.1:8000/admin/taggit/tag را در مرورگر خود باز کنید. صفحه سرپرست را با لیستی از اشیاء Tag از برنامه taggit مشاهده خواهید کرد:



Action: 0 of 3 selected

	NAME	SLUG
<input type="checkbox"/>	django	django
<input type="checkbox"/>	jazz	jazz
<input type="checkbox"/>	music	music

[3 Tags](#)

به <http://127.0.0.1:8000/admin/blog/post> بروید و برای ویرایش آن روی یک پست کلیک کنید. خواهید دید که ارسال‌ها اکنون دارای یک قسمت جدید برچسب‌ها به شرح زیر است، که می‌توانید برچسب‌ها را به راحتی ویرایش کنید:

Tags:

jazz, music

A comma-separated list of tags.

اکنون، ما برای نمایش برچسب‌ها، پست‌های وبلاگ شما را ویرایش خواهیم کرد. الگوی وبلاگ /post /list.html را باز کنید و کد HTML زیر عنوان زیر را اضافه کنید:

```
<p class="tags">Tags: {{ post.tags.all|join:", " }}</p>
```

فیلتر قالب پیوند به عنوان روش پیوند رشته (Python join) کار می کند تا عناصر را با رشته داده شده به هم بباورد. /ra در مرورگر خود باز کنید. شما باید بتوانید لیست برچسب ها را در زیر هر عنوان مشاهده کنید:

Who was Django Reinhardt?

Tags: jazz, music

Published Dec. 14, 2017, 8:54 a.m. by admin

اکنون ، ما نمای post_list خود را ویرایش خواهیم کرد تا به کاربران اجازه دهیم همه نوشه های دارای برچسب مشخص را لیست کنند. پرونده views.py برنامه وبلاگ خود را باز کنید ، فرم مدل Django-taggit Tag را وارد کنید و نمایش thepost_list را تغییر دهید تا پست های اختیاری را با یک برچسب فیلتر کنید ، به شرح زیر:

```
from taggit.models import Tag

def post_list(request, tag_slug=None):
    object_list = Post.published.all()
```

```
tag = None

if tag_slug:
    tag = get_object_or_404(Tag, slug=tag_slug)
    object_list = object_list.filter(tags__in=[tag])

paginator = Paginator(object_list, 3) # 3 posts in each page
# ...
```

نمای post_list اکنون به شرح زیر عمل می کند:

1. یک پارامتر tag_slug اختیاری می گیرد که دارای مقدار پیش فرض None است. این پارامتر در URL وارد می شود.
2. در داخل نمای QuerySet اولیه را ایجاد می کنیم ، کلیه پست های منتشر شده را بازیابی می کنیم و در صورت وجود یک شلخته برچسب مشخص ، با استفاده از میانبر get_object_or_404 () شیء Tag را با استفاده از گنجشک داده شده دریافت می کنیم.
3. سپس ، لیست ارسال ها را توسط پستهایی که حاوی برچسب داده شده هستند ، فیلتر می کنیم. از آنجا که این یک رابطه بسیار زیاد است ، ما باید برچسب های موجود در یک لیست مشخص را فیلتر کنیم ، که در مورد ما ، فقط یک عنصر دارد.

به یاد داشته باشید که QuerySets برای بازیابی پست ها فقط هنگام ارزیابی می شوند که ما هنگام ارائه الگو ، لیست را حل کنیم.

در آخر ، عملکرد render() را در پایین نمای اصلاح کنید تا متغیر برچسب به الگو منتقل شود. سرانجام این نمای باید به شکل زیر باشد:

```
def post_list(request, tag_slug=None):
    object_list = Post.published.all()
    tag = None

    if tag_slug:
        tag = get_object_or_404(Tag, slug=tag_slug)
        object_list = object_list.filter(tags__in=[tag])
```

```
paginator = Paginator(object_list, 3) # 3 posts in each page
page = request.GET.get('page')
try:
    posts = paginator.page(page)
except PageNotAnInteger:
    # If page is not an integer deliver the first page
    posts = paginator.page(1)
except EmptyPage:
    # If page is out of range deliver last page of results
    posts = paginator.page(paginator.num_pages)
return render(request, 'blog/post/list.html', {'page': page,
                                                'posts': posts,
                                                'tag': tag})
```

پرونده urls.py برنامه وبلاگ خود را باز کنید ، از الگوی URL PostListView URL مستقر در کلاس اظهار نظر کنید و نمای post_list را مانند این اظهار کنید:

```
path('', views.post_list, name='post_list'),
# path('', views.PostListView.as_view(), name='post_list'),
```

الگوی URL زیر را اضافه کنید تا پست ها را با برچسب لیست کنید:

```
path('tag/<slug:tag_slug>/',
      views.post_list, name='post_list_by_tag'),
```

همانطور که مشاهده می کنید ، هر دو الگوی به یک منظر اشاره می کنند ، اما ما آنها را متفاوت می نامیم. الگوی اول نمای post_list را بدون هیچ گونه پارامتر اختیاری فراخوانی می کند ، در حالی که الگوی دوم با پارامتر tag_slug با نمای تماس می گیرد. برای تطابق پارامتر به عنوان یک رشته کوچک با حروف یا اعداد ASCII ، به علاوه کarakترهای hyphen و underscore از مبدل مسیر slug استفاده می کنیم.

از آنجا که ما از نمای post_list استفاده می کنیم blog/post/list.html الگو را ویرایش کنید و صفحه بندی را برای استفاده از موضوع پست اصلاح کنید:

```
{% include "pagination.html" with page=posts %}
```

خطوط زیر را بالای حلقه { } اضافه کنید:

```
{% if tag %}
  <h2>Posts tagged with "{{ tag.name }}</h2>
{% endif %}
```

اگر کاربر به وبلاگ دسترسی پیدا کند ، لیست تمام ارسال ها را مشاهده می کند. اگر آنها بر اساس برچسب هایی با برچسب خاص فیلتر شوند ، برچسب هایی را که توسط آنها فیلتر شده است مشاهده می کنند. اکنون نحوه نمایش برچسب ها را به شرح زیر تغییر دهید:

```
<p class="tags">
    Tags:
    {% for tag in post.tags.all %}
        <a href="{% url "blog:post_list_by_tag" tag.slug %}">
            {{ tag.name }}
        </a>
        {% if not forloop.last %}, {% endif %}
    {% endfor %}
</p>
```

اکنون ، ما تمام برچسب های یک پست را نشان می دهیم که یک لینک سفارشی به URL نشان می دهد تا پست های آن برچسب را فیلتر کنند. ما URL را با وبلاگ { URL "blog: post_list_by_tag" tag.slug \% } با استفاده از نام URL و برچسب slug به عنوان پارامتر آن می سازیم. ما برچسب ها را با کاما از هم جدا می کنیم.

در مرورگر خود `http://127.0.0.1:8000/blog` را باز کنید و بر روی هر نشانگر کلیک کنید. لیست پست های فیلتر شده با آن برچسب را مانند این مشاهده خواهید کرد:

My Blog

Posts tagged with "jazz"

Who was Django Reinhardt?

Tags: [jazz](#) , [music](#)

Published Dec. 14, 2017, 8:54 a.m. by admin

Who was Django Reinhardt.

Page 1 of 1.

بازیابی پست ها با تشابه

اکنون که برچسب گذاری را برای پست های و بلاگ خود اجرا کرده ایم ، می توانیم کارهای جالب بسیاری را با آنها انجام دهیم. با استفاده از برچسب ها ، می توانیم پست های و بلاگ خود را بسیار خوب طبقه بندی کنیم. ارسالها درباره موضوعات مشابه ، چندین برچسب مشترک دارند. ما برای نمایش پست های مشابه با تعداد برچسب هایی که به اشتراک می گذارند ، عملکردی ایجاد خواهیم کرد. به این ترتیب ، وقتی کاربر پستی را می خواند ، می توانیم به آنها پیشنهاد کنیم که سایر پست های مرتبط را بخوانند.

برای بازیابی پست های مشابه برای یک پست خاص ، باید مراحل زیر را انجام دهیم:

1. کلیه برچسب ها را برای پست فعلی بازیابی کنید
2. دریافت تمام پست هایی که با هر یک از آن برچسب ها برچسب خورده است
3. برای جلوگیری از توصیه همان پست ، پست فعلی را از آن لیست خارج کنید
- 4- نتایج را بر اساس تعدادی از برچسب های به اشتراک گذاشته شده با پست فعلی سفارش دهید
- 5- در صورت دو یا چند پست با همان تعداد برچسب ، جدیدترین پست را توصیه کنید
6. پرس و جو را به تعداد پستهایی که می خواهیم پیشنهاد کنیم محدود کنید

این مراحل به یک QuerySet پیچیده ترجمه می شود که در نمای post_detail ما قرار خواهیم داد. پرونده views.py برنامه و بلاگ خود را باز کنید و واردات زیر را در بالای آن اضافه کنید:

```
from django.db.models import Count
```

این تابع شمارش تعداد عملکرد Django ORM است که به ما امکان می دهد تعداد برچسب ها را انجام دهیم.

تابع django.db.models شامل توابع جمع آوری زیر است:

میانگین: میانگین ارزش

حداکثر: حداکثر مقدار

حداقل: حداقل مقدار

تعداد: اشیاء شمارش می کنند

می توانید در مورد تجمعی در <https://docs.djangoproject.com/en/2.0/topics/db/aggregation/> بیاموزید.

قبل از عملکرد رندر () ، با همان سطح تورفتگی ، خطوط زیر را در نمای post_detail اضافه کنید:

```
# List of similar posts
post_tags_ids = post.tags.values_list('id', flat=True)
similar_posts = Post.published.filter(tags__in=post_tags_ids) \
    .exclude(id=post.id)
similar_posts = similar_posts.annotate(same_tags=Count('tags')) \
    .order_by('-same_tags', '-publish')[:4]
```

کد قبلی به شرح زیر است:

1. ما یک لیست از شناسه های پایتون را برای برچسب های پست فعلی بازیابی می کنیم، QuerySet value_list() نوارهای را با مقادیر مربوط به قسمتهای داده شده بر می گرداند. برای بدست آوردن یک لیست صاف مانند [1 ، 2 ، 3 ، ...] صاف = صحیح است.

2. همه پستهایی را که حاوی هر یک از این برچسب ها است ، دریافت می کنیم ، به استثنای خود پست فعلی.

3. ما از توابع جمع برای تولید یک فیلد محاسبه شده — same_tags — استفاده می کنیم که شامل تعدادی از برچسب های به اشتراک گذاشته شده با تمام برچسب های پرسیده شده است.

4- نتیجه را با توجه به تعدادی از برچسب های مشترک (مرتبه نزولی) سفارش می دهیم و با انتشار این مطلب ، پست های اخیر را برای اولین بار برای پست ها با همان برچسب مشترک نمایش می دهیم ما نتیجه را قطعه قطعه می کنیم تا فقط چهار پست اول را بازیابی کنیم.

برای تابع ()؛ شی similar_posts را به فرهنگ لغت متن اضافه کنید:

```
return render(request,
    'blog/post/detail.html',
    {'post': post,
     'comments': comments,
     'new_comment': new_comment,
     'comment_form': comment_form,
     'similar_posts': similar_posts})
```

اکنون الگوی blog/post/detail.html را ویرایش کنید و قبل از لیست نظرات توضیحات ، کد زیر را اضافه کنید:

```
<h2>Similar posts</h2>
{% for post in similar_posts %}
<p>
  <a href="{{ post.get_absolute_url }}>{{ post.title }}</a>
</p>
{% empty %}
  There are no similar posts yet.
{% endfor %}
```

اکنون ، صفحه جزئیات پست شما باید به صورت زیر باشد:

Who was Django Reinhardt?

Published Dec. 14, 2017, 8:54 a.m. by admin

Who was Django Reinhardt.

[Share this post](#)

Similar posts

[Miles Davis favourite songs](#)

[Notes on Duke Ellington](#)

اکنون می توانید پست های مشابه را با موفقیت به کاربران خود توصیه کنید. Django-taggit همچنین شامل یک مدیر similar_objects است که می توانید برای بازیابی اشیاء با برچسب های مشترک استفاده کنید. می توانید در تمام مدیران سایت Django- Taggit <https://django-aggit.readthedocs.io/fa/latest/api.html> جستجو کنید.

همچنین می توانید لیست برچسب ها را به همان روشی که در الگوی blog/post/list.html انجام دادید به الگوی جزئیات ارسال خود اضافه کنید.

خلاصه

در این فصل یادگرفتید که چگونه با فرم های جنگو و فرم های مدل کار کنید. شما سیستمی ایجاد کرده اید تا محتوای سایت خود را از طریق ایمیل به اشتراک بگذارید و یک سیستم نظر برای وبلاگ خود ایجاد کنید. شما برچسب گذاری به پست های وبلاگ خود ، ادغام یک برنامه قابل استفاده مجدد ، و QuerySets پیچیده ای را برای بازیابی اشیاء با شباهت ایجاد کرده اید.

در فصل بعدی یاد می گیرید که چگونه برحسب ها و فیلترهای قالب دلخواه را ایجاد کنید. همچنین می توانید یک نقشه سایت سفارشی برای پست های و بلاگ خود بسازید و عملکرد جستجوی متن کامل را برای پست های و بلاگ خود پیاده سازی کنید.

فصل سوم

برنامه و بلاگ خود را گسترش

دهید

فصل قبل اصول فرم را طی کرد و شما یاد گرفتید که چگونه برنامه های شخص ثالث را در پروژه خود ادغام کنید.

این فصل نکات زیر را در بر می گیرد:

ایجاد برچسب ها و فیلترهای قالب های سفارشی

افزودن نقشه سایت و فید پست

پیاده سازی جستجوی متن کامل با PostgreSQL

ایجاد برچسب ها و فیلترهای قالب های سفارشی

جنگو انواع برچسب های داخلی داخلی را ارائه می دهد ، مانند { %if% } یا { %block% }. شما در الگوهای خود از چندین مورد استفاده کرده اید. شما می توانید یکمرجع کامل برچسب ها و فیلترهای قالب داخلی در <https://docs.djangoproject.com/fa/2.0/ref/templates/builtins>

با این حال ، Django همچنانی به شما امکان می دهد تا برچسب های الگوی خود را برای انجام اقدامات سفارشی ایجاد کنید. برچسب های قالب سفارشی هنگامی که شما نیاز به اضافه کردن قابلیت هایی به قالب های خود دارید که توسط مجموعه اصلی برچسب های قالب Django پوشیده نیست ، بسیار مفید است.

ایجاد برچسب های قالب سفارشی

Django توابع کمکی زیر را ارائه می دهد که به شما امکان می دهد برچسب های الگوی خود را با روشی آسان ایجاد کنید: `Simple_tag`: داده ها را پردازش می کند و یک رشته را بر می گرداند

:inclusive_tag داده ها را پردازش کرده و یک الگوی ارائه شده را برمی گرداند

برچسب های قالب باید در برنامه های جنگو زندگی کنند.

در داخل فهرست برنامه وبلاگ خود ، یک فهرست جدید ایجاد کنید ، آن را برچسب های الگوبرداری کنید و یک پرونده خالی __init__.py را به آن اضافه کنید. یک پرونده دیگر را در همان پوشه ایجاد کرده و آنرا blog_tags.py بنامید.

ساختار پرونده برنامه وبلاگ باید به شرح زیر باشد:

```
blog/
    __init__.py
    models.py
    ...
    templatetags/
        __init__.py
        blog_tags.py
```

نحوه نامگذاری پرونده مهم است. برای بارگذاری برچسب ها در قالب ها ، از نام این ماثول استفاده خواهد کرد.

ما با بازیابی یک برچسب ساده برای بازیابی کل پست های منتشر شده در وبلاگ شروع خواهیم کرد. پرونده blog_tags.py را که درست کردید ویرایش کنید و کد زیر را اضافه کنید:

```
from django import template
from ..models import Post

register = template.Library()

@register.simple_tag
def total_posts():
    return Post.published.count()
```

ما یک برچسب الگوی ساده ایجاد کرده ایم که تعداد پست های منتشر شده تاکنون را بر می گرداند. هر ماثول برچسب قالب باید حاوی آن باشد

متغیر به نام Register برای داشتن یک کتابخانه برچسب معتبر است ، این متغیر نمونه ای از یک کتابخانه الگوی است و از آن برای ثبت برچسب ها و فیلترهای قالب خاص ما استفاده می شود. سپس یک تگ به نام total_posts را با یک تابع Python تعریف می کنیم و از دکوراتور @register.simple_tag برای ثبت تابع به عنوان برچسب ساده استفاده می کنیم. Django از نام تابع به عنوان نام تگ استفاده می کند. اگر می خواهید با استفاده از نام دیگری آن را ثبت کنید ، می توانید این کار را با مشخص کردن یک ویژگی اسمی مانند @register.simple_tag(name='my_tag') انجام دهید.

قبل از استفاده از برچسب های قالب سفارشی ، شما باید آنها را با استفاده از برچسب { % load blog_tags % } برای الگو در دسترس قرار دهید. همانطور که قبل از نیز گفته شد ، باید از نام ماثول پایتون که حاوی برچسب ها و فیلترهای قالب شماست ، استفاده کنید. قالب html/base.html را باز کنید و { % load blog_tags % } را در بالای آن اضافه کنید تا ماثول برچسب های قالب خود بارگیری شود. سپس از برچسب ایجاد شده برای نمایش کل پست های خود استفاده کنید. فقط { % total_posts % } را به الگوی خود اضافه کنید. سرانجام این الگو باید به صورت زیر باشد:

```
{% load blog_tags %}
{% load static %}
<!DOCTYPE html>
<html>
<head>
    <title>{% block title %}{% endblock %}</title>
    <link href="{% static "css/blog.css" %}" rel="stylesheet">
</head>
```

```
<body>
    <div id="content">
        {% block content %}
        {% endblock %}
    </div>
    <div id="sidebar">
        <h2>My blog</h2>
        <p>This is my blog. I've written {% total_posts %} posts so far.</p>
    </div>
</body>
</html>
```

برای پیگیری پرونده های جدید اضافه شده به پروژه ، باید سرور را مجدداً راه اندازی کنیم. سرور توسعه را با Ctrl + C متوقف کرده و مجدداً آن را با استفاده از دستور زیر اجرا کنید:

```
python manage.py runserver
```

/blog را در مرورگر خود باز کنید. شما باید بینید

تعداد کل ارسال ها در نوار کناری سایت به شرح زیر است:

My blog

This is my blog. I've written 4 posts so far.

قدرت برچسب های قالب سفارشی این است که می توانید هر داده ای را پردازش کرده و بدون توجه به نمای اجرا شده ، آن را به هر قالب اضافه کنید. می توانید QuerySets را انجام داده یا داده ها را پردازش کنید تا نتایج را در قالب های خود نمایش دهید.

اکنون ، ما برای نمایش آخرین ارسال ها در نوار کناری و بلاگ خود ، یک برچسب دیگر ایجاد خواهیم کرد. این بار از برچسب گنجاندن استفاده خواهیم کرد. با استفاده از یک برچسب گنجاندن ، می توانید یک الگوی با متغیرهای زمینه برگردانده شده توسط برچسب الگوی خود ارائه دهید. blog_tags.pyfile را ویرایش کنید و کد زیر را اضافه کنید:

```
@register.inclusion_tag('blog/post/latest_posts.html')
```

```
def show_latest_posts(count=5):
    latest_posts = Post.published.order_by('-publish')[ :count]
    return {'latest_posts': latest_posts}
```

در کد قبلی ، برچسب قالب را با استفاده از @register.inclusion_tag ثبت می کنیم و الگوی را که باید با مقادیر برگشت یافته با استفاده از log / post / last_posts.html ارائه شود ، مشخص می کنیم. برچسب قالب ما یک پارامتر شمارش اختیاری را انتخاب می کند که به صورت پیش فرض در 5. این پارامتر به ما امکان می دهد تعداد پستهایی را که می خواهیم نمایش دهیم را مشخص کنیم. ما از این متغیر برای محدود کردن نتایج پرس و جو

```
.Post.published.order_by('-publish')[ : count]
```

استفاده می کنیم [: count]. توجه داشته باشید که این تابع به جای یک مقدار ساده ، فرهنگ لغت متغیرها را بر می گردد. برچسب های گنجاندن باید یک فرهنگ لغت از مقادیر را برگردانند ، که به عنوان زمینه برای ارائه الگوی مشخص شده استفاده می شود. برچسب الگوی که ما فقط ایجاد کردیم به شما امکان می دهد تعداد پست های اختیاری را برای نمایش به عنوان { % show_latest_posts 3 % }. مشخص کنید.

اکنون یک فایل الگوی جدید را در زیر blog/post/ایجاد کنید و نام آن را last_posts.html بگذارید. کد زیر را به آن اضافه کنید:

```
<ul>
{% for post in latest_posts %}
<li>
  <a href="{{ post.get_absolute_url }}>{{ post.title }}</a>
</li>
{% endfor %}
</ul>
```

در کد قبلی ، ما با استفاده از متغیر `last_posts` که توسط برچسب الگوی ما برگردانده شده است ، لیست نامرتب از پست ها را نمایش می دهیم. اکنون الگوی `blog / base.html` را ویرایش کرده و برچسب الگوی جدید را برای نمایش سه پست آخر اضافه کنید. کد نوار کناری باید به شکل زیر باشد:

```
<div id="sidebar">
  <h2>My blog</h2>
  <p>This is my blog. I've written {% total_posts %} posts so far.</p>

  <h3>Latest posts</h3>
  {% show_latest_posts 3 %}
</div>
```

برچسب قالب به عبور تعداد ارسال ها برای نمایش داده می شود و قالب در متن با متن مشخص ارائه می شود. اکنون به مرورگر خود برگردید و صفحه را تازه کنید. نوار کناری اکنون باید مانند این باشد:

My blog

This is my blog. I've written 4 posts so far.

Latest posts

- Miles Davis favourite songs
- Notes on Duke Ellington
- Another post

سرانجام ، ما یک برجسب الگوی ساده ایجاد خواهیم کرد که نتیجه را در یک متغیر ذخیره می کند و می توان از آن استفاده کرد نه اینکه مستقیماً از آن استفاده کند. ما یک برجسب برای نمایش بیشترین اظهار نظرها ایجاد خواهیم کرد. پرونده blog_tags.py را ویرایش کنید و برجسب واردات و الگوی زیر را در آن اضافه کنید:

```
from django.db.models import Count

@register.simple_tag
def get_most_commented_posts(count=5):
    return Post.published.annotate(
        total_comments=Count('comments')
    ).order_by('-total_comments')[:count]
```

در برجسب قالب قبلی ، ما با استفاده از تابع annotate() برای ایجاد تعداد کل نظرات برای هر پست ، یک QuerySet ایجاد می کنیم. ما از تابع Count جمع استفاده می کنیم تا تعداد نظرات را در قسمت محاسبه شده total_comments برای هر شیء پست ذخیره کنیم. ما QuerySet را با قسمت محاسبه شده به ترتیب نزولی سفارش می دهیم. ما همچنین یک متغیر شمارش اختیاری را برای محدود کردن تعداد کل اشیاء برگشت یافته فراهم می کنیم.

علاوه بر تعداد ، `Sum`، `Max` و `Avg` توابع جمع آوری می دهد. می توانید اطلاعات بیشتر در مورد عملکردهای جمع آوری را در <https://docs.djangoproject.com/fa/2.0/topics/db/aggregation> بخوانید.

الگوی `blog` را ویرایش کنید و کد زیر را در عنصر نوار کناری `<div>` اضافه کنید:

```
<h3>Most commented posts</h3>
{% get_most_commented_posts as most_commented_posts %}
<ul>
{% for post in most_commented_posts %}
    <li>
        <a href="{{ post.get_absolute_url }}>{{ post.title }}</a>
    </li>
{% endfor %}
</ul>
```

ما نتیجه را در یک متغیر دلخواه با استفاده از آرگومان به دنبال نام متغیر ذخیره می کنیم. برای برچسب الگوی خود ، از `{% get_most_commented_posts as most_commented_posts %}` استفاده می کنیم تا نتیجه برچسب قالب را در یک متغیر جدید با نام `most_commented_posts` ذخیره کنیم. سپس ، بازگردانی را که دارای یک لیست ناموزون است ، نمایش می دهیم. اکنون مرورگر خود را باز کرده و صفحه را تازه کنید تا نتیجه نهایی را ببینید.

باید مانند زیر باشد:

The screenshot shows a web browser window with a light gray header bar. The address bar contains the URL "127.0.0.1:3000/blog". The main content area displays a blog titled "My Blog". There are two posts listed:

- Miles Davis favourite songs**
Tags: [jazz](#), [music](#)
Published Dec. 14, 2017, 10:59 p.m. by admin
Miles Dewey Davis III was an American jazz trumpeter, bandleader, and composer.
- Notes on Duke Ellington**
Tags: [jazz](#), [music](#)
Published Dec. 14, 2017, 9:59 p.m. by admin
Edward Kennedy "Duke" Ellington was an American composer, pianist, and bandleader of a jazz orchestra.

To the right of the posts is a sidebar with the following sections:

- My blog**
This is my blog. I've written 4 posts so far.
- Latest posts**
 - [Miles Davis favourite songs](#)
 - [Notes on Duke Ellington](#)
 - [Another post](#)
- Most commented posts**
 - [Notes on Duke Ellington](#)
 - [Who was Django Reinhardt?](#)
 - [Another post](#)
 - [Miles Davis favourite songs](#)

اکنون درباره روش ساخت برچسب های قالب های سفارشی ایده روشنی دارید.

می توانید اطلاعات بیشتر در مورد آنها را در اینجا بخوانید

[./https://docs.djangoproject.com/en/2.0/howto/custom-template-tags](https://docs.djangoproject.com/en/2.0/howto/custom-template-tags)

ایجاد فیلترهای قالب سفارشی

جنگو انواع فیلترهای داخلی دارکه به شما امکان می دهد متغیرها را در قالب ها تغییر دهید. این توابع پایتون هستند که یک یا دو پارامتر را در بر می گیرند - مقدار متغیر مورد استفاده برای آن و یک آرگومان اختیاری. آنها مقداری را که با یک فیلتر دیگر نمایش داده می شود یا درمان می کنند، بر می گردانند.

یک فیلتر به مانند `{variable|my_filter:"foo"}`. فیلترهای دارای آرگومان شبیه `{variable|my_filter:"foo"}` هستند. به عنوان مثال می توانید به همان اندازه فیلترهای مورد نظر خود را اعمال کنید، `{variable|filter1|filter2}`، و هر کدام از آنها بر روی خروجی تولید شده توسط فیلتر قبلی اعمال می شود.

ما یک فیلتر سفارشی ایجاد خواهیم کرد تا بتوانیم در پست های وبلاگ خود از نحو نشانه گذاری استفاده کرده و سپس محتوای آنها را در قالب ها به HTML تبدیل کنیم. Markdown یک نحو ساده قالب بندی متن است که کاربرد آن بسیار ساده است و برای تبدیل شدن به HTML در نظر گرفته شده است. می توانید اصول اولیه این قالب را در

<https://daringfireball.net/projects/markdown/basics>

ابتدا با استفاده از دستور زیر، ماژول علامت گذاری Python را از طریق پیپ نصب کنید:

```
pip install Markdown==2.6.11
```

سپس پرونده `blog_tags.py` را ویرایش کنید و کد زیر را وارد کنید:

```
from django.utils.safestring import mark_safe
import markdown

@register.filter(name='markdown')
def markdown_format(text):
    return mark_safe(markdown.markdown(text))
```

ما فیلترهای قالب را به همان روش و برچسب های قالب ثبت می کنیم. برای جلوگیری از برخورد بین نام عملکرد و ماژول علامت گذاری، عملکرد خود را `markdown_format` نامیم و علامت گذاری فیلتر را برای استفاده در قالب ها، از جمله `{variable|markdown}` نامیم. Django از کد HTML تولید شده توسط فیلترها فرار می کند. ما از تابع `mark_safe` ارائه شده توسط Django برای نشان دادن نتیجه به عنوان HTML این برای ارائه در الگو استفاده می کنیم. به طور پیش فرض، جنگو به هیچ کد HTML اعتماد نخواهد کرد و قبل از قرار دادن در خروجی، از آن فرار خواهد کرد تنها

استثناء متغیرهایی هستند که به عنوان اینم از فرار علامت گذاری می شوند این رفتار مانع از خروج جنگو به طور بالقوه خطرناک می شود.

HTML و به شما امکان می دهد استثنائی برای بازگشت HTML اینم ایجاد کنید.
اکنون مژول برجسب های قالب خود را در لیست ارسال و الگوهای جزئیات بارگذاری کنید. خط زیر را در بالای وبلاگ /
اضافه کنید post / list.html

وقالب های blog/post/detail.html سپس از برجسب extends %} :

```
{% load blog_tags %}
```

در قالب های post / detail.html ، به خط زیر نگاهی بیندازید:

```
{{ post.body|linebreaks }}
```

آن را با موارد زیر جایگزین کنید:

```
{{ post.body|markdown }}
```

سپس در پرونده post / list.html خط زیر را جایگزین کنید:

```
{{ post.body|truncatewords:30|linebreaks }}
```

سپس آن را با موارد زیر مبادله کنید:

```
{{ post.body|markdown|truncatewords_html:30 }}
```

فیلتر `truncatewords_html` یک رشته را پس از تعداد مشخصی از کلمات کوتاه می‌کند و از برچسب‌های غیرمجاز جلوگیری می‌کند.

اکنون در مرورگر خود `http://127.0.0.1:8000/admin/blog/post/add` را باز کنید و به `body` زیر پستی اضافه کنید:

```
This is a post formatted with markdown
-----
*This is emphasized* and **this is more emphasized**.

Here is a list:

* One
* Two
* Three

And a [link to the Django website](https://www.djangoproject.com/)
```

مرورگر خود را باز کنید و به نحوه ارائه پست نگاه بیندازید. باید خروجی زیر را مشاهده کنید:

Markdown post

Published Dec. 15, 2017, 8:42 a.m. by admin

This is a post formatted with markdown

This is emphasized and this is more emphasized.

Here is a list:

- One
- Two
- Three

And a [link to the Django website](#)

همانطور که در تصویر قبلی مشاهده می کنید ، فیلترهای قالب های سفارشی برای شخصی سازی قالب بندی بسیار مفید هستند. می توانید اطلاعات بیشتر در مورد فیلترهای سفارشی را در اینجا پیدا کنید

<https://docs.djangoproject.com/fa/2.0/howto/custom-template-tags/#writing-customtemplate-filters>

[نقشه سایت را به سایت خود اضافه کنید](#)

جنگو با یک چارچوب نقشه سایت ، که به شما امکان می دهد نقشه سایت های سایت خود را بصورت دینامیک تولید کنید. نقشه سایت یک فایل XML است که به موتورهای جستجو صفحات وب سایت شما ، ارتباط آنها و اینکه چه تعداد مرتبه به روز می شوند ، می گوید. با استفاده از نقشه سایت ، به خزنه هایی که محتوای وب سایت شما را فهرست می کنند ، کمک می کنید.

چارچوب نقشه سایت Django به شما امکان می دهد اشیاء را به وب سایت های خاصی که با پروژه شما در حال اجرا هستند مرتبط کنید. وقتی می خواهید چندین سایت را با استفاده از یک پروژه تک جنگو انجام دهید ، این کار مفید خواهد بود. برای نصب چارچوب نقشه سایت ، باید هم سایتها و هم برنامه های نقشه سایت را در پروژه خود فعال کنید. پرونده تنظیمات.py پروژه خود را ویرایش کنید و django.contrib.sites INSTALLED_APPS را به تنظیمات django.contrib.sitemaps اضافه کنید. همچنین ، یک تنظیم جدید برای شناسه سایت به شرح زیر تعریف کنید:

```
SITE_ID = 1

# Application definition
INSTALLED_APPS = [
    # ...
    'django.contrib.sites',
    'django.contrib.sitemaps',
]
```

اکنون ، دستور زیر را برای ایجاد جداول برنامه سایت Django در پایگاه داده اجرا کنید:

```
python manage.py migrate
```

باید خروجی را مشاهده کنید که شامل خطوط زیر باشد:

```
Applying sites.0001_initial... OK
Applying sites.0002_alter_domain_unique... OK
```

برنامه سایت اکنون با بانک اطلاعات همکام شده است. اکنون یک پرونده جدید را در فهرست برنامه و بلاگ خود ایجاد کرده و آن را sitemaps.py بنامید. پرونده را باز کنید و کد زیر را به آن اضافه کنید

```
from django.contrib.sitemaps import Sitemap
from .models import Post

class PostSitemap(Sitemap):
    changefreq = 'weekly'
    priority = 0.9

    def items(self):
        return Post.published.all()

    def lastmod(self, obj):
        return obj.updated
```

ما با میراث کلاس نقشه سایت مازول نقشه سایت ، یک نقشه سایت اختصاصی ایجاد می کنیم. تغییر freq و ویژگی های اولویت نشان دهنده فرکانس تغییر صفحات پست شما و ارتباط آنها در وب سایت شما (حداکثر مقدار 1) است. روش اقلام () اشیاء را در این نقشه سایت باز می گرداند. به طور پیش فرض ، Django برای بازیابی آدرس اینترنتی خود از متد url () برای هر شی استفاده می کند.

به یاد داشته باشید که ما این روش را در فصل 1 ، ایجاد یک برنامه وبلاگ ایجاد کرده ایم تا URL متعارف برای پست ها را بازیابی کنیم. اگر می خواهید URL را برای هر شی مشخص کنید ، می توانید یک روش موقعیت مکانی را به کلاس نقشه سایت خود اضافه کنید. آخرین روش mod هر شیئی را که توسط آیتم ها برگشت داده می شود دریافت می کند و آخرین باری که شیء تغییر یافته است بر می گرداند. هر دو روش freq و اولویت را تغییر می دهند یا می توانند روش یا ویژگی باشند. می توانید به مرجع کامل نقشه سایت در اسناد رسمی جنگو واقع در <https://docs.djangoproject.com/en/2.0/ref/contrib/sitemaps/>.

سرانجام ، فقط باید URL نقشه سایت خود را اضافه کنید. پرونده اصلی urls.py پروژه خود را ویرایش کرده و نقشه سایت را به شرح زیر اضافه کنید:

```
from django.urls import path, include
from django.contrib import admin
from django.contrib.sitemaps.views import sitemap
from blog.sitemaps import PostSitemap

sitemaps = {
    'posts': PostSitemap,
}

urlpatterns = [
    path('admin/', admin.site.urls),
    path('blog/', include('blog.urls', namespace='blog')),
    path('sitemap.xml', sitemap, {'sitemaps': sitemaps},
         name='django.contrib.sitemaps.views.sitemap')
]
```

در کد قبلی ، واردات لازم را درج کردیم و فرهنگ لغت نقشه سایت را تعریف کردیم. ما یک الگوی URL تعریف کردیم که با مطابقت دارد و از نمای نقشه سایت استفاده می کند. فرهنگ لغت نقشه سایت به نمای نقشه سایت منتقل می شود. اکنون سرور توسعه را اجرا کنید و <http://127.0.0.1:8000/sitemap.xml> را در مرورگر خود باز کنید. به خروجی XML زیر توجه کنید:

```
<?xml version="1.0" encoding="utf-8"?>
<urlset xmlns="http://www.sitemaps.org/schemas/sitemap/0.9">
  <url>
    <loc>http://example.com/blog/2017/12/15/markdown-post/</loc>
    <lastmod>2017-12-15</lastmod>
    <changefreq>weekly</changefreq>
    <priority>0.9</priority>
  </url>
  <url>
    <loc>
      http://example.com/blog/2017/12/14/who-was-django-reinhardt/
    </loc>
    <lastmod>2017-12-14</lastmod>
    <changefreq>weekly</changefreq>
    <priority>0.9</priority>
  </url>
</urlset>
```

URL برای هر پست با استفاده از روش `get_absolute_url()` ساخته شده است.

ویژگی lastmod مطابق با ما در قسمت تاریخ به روز شده مطابقت دارد ، همانطور که در نقشه سایت ما مشخص کردیم ، و
ویژگی های تغییر freq و اولویت نیز از کلاس PostSitemap ماگرفته شده است. می توانید ببینید که دامنه مورد استفاده
برای ساختن URL ها shembull.com است. این دامنه از یک موضوع Site ذخیره شده در پایگاه داده می شود. این شیء
پیش فرض زمانی ایجاد شده است که چارچوب سایت را با پایگاه داده خود همگام می کیم.

را در مرورگر خود باز کنید. شما باید چیزی شبیه به این را ببینید:

Select site to change

ADD SITE

Action: ----- Go 0 of 1 selected

DOMAIN NAME	DISPLAY NAME
example.com	example.com

1 site

تصویر قبلی شامل نمای سرپرست نمایش صفحه برای چارچوب سایت است. در اینجا می توانید دامنه یا هاست را تنظیم کنید که توسط چارچوب سایت و برنامه های کاربردی وابسته به آن مورد استفاده قرار گیرد.

برای تولید URL های که در محیط محلی ما وجود دارند ، نام دامنه را به localhost/8000 تغییر دهید: ، همانطور که در شکل زیر نشان داده شده است

screenshot, and save it:

Change site

Domain name:	localhost:8000
Display name:	localhost:8000

URL های نمایش داده شده در فید شما اکنون با استفاده از این نام میزبان ساخته می شوند. در یک محیط تولید ، شما باید از نام دامنه خود برای چارچوب سایت استفاده کنید.

ایجاد فید برای پست های وبلاگ خود

دارای یک چارچوب داخلی در زمینه ایجاد سندیکا است که می توانید با استفاده از چارچوب سایت ، نقشه های RSS یا Atom را بصورت پویا تولید کنید. یک فید وب یک قالب داده (معمولًا XML) است که محتوای مرتبًا به روز شده را در اختیار کاربران قرار می دهد. کاربران قادر خواهند بود با استفاده از یک جمع کننده فید ، نرم افزاری که برای خواندن فیدها استفاده می شود و اعلان های محتوای جدیدی را دریافت کنید ، در فید شما مشترک شوید.

یک پرونده جدید را در فهرست برنامه وبلاگ خود ایجاد کنید و نام آن را feeds.py بگذارید. خطوط زیر را به آن اضافه کنید:

```
from django.contrib.syndication.views import Feed
from django.template.defaultfilters import truncatewords
from .models import Post

class LatestPostsFeed(Feed):
    title = 'My blog'
    link = '/blog/'
    description = 'New posts of my blog.'

    def items(self):
        return Post.published.all()[:5]

    def item_title(self, item):
        return item.title

    def item_description(self, item):
        return truncatewords(item.body, 30)
```

ابتدا کلاس Feed از چارچوب syndication را طبقه بندی می کنیم. عناوین عنوان ، پیوند و توضیحات به ترتیب با عناوین <link> و <description> RSS مطابقت دارند.

روش items() اشیاء را که باید در فید گنجانده شوند بازیابی می کند.

ما فقط پنج پست آخر منتشر شده برای این فید را بازیابی می کنیم. متدهای item_title() و item_description() هر شیئی را که توسط اقلام برگشت داده می شوند دریافت می کنند و عنوان و توضیحات مربوط به هر مورد را برمی گردانند. ما از کلمات کوتاه ساخته شده در فیلتر قالب برای ساخت توضیحات پست و بلاگ با 30 کلمه اول استفاده می کنیم.

اکنون پرونده blog/urls.py را ویرایش کنید ، آخرین Posts را که اخیراً ایجاد کرده اید وارد کنید و فید را در یک الگوی URL جدید فوراً وارد کنید:

```
from .feeds import LatestPostsFeed

urlpatterns = [
    # ...
    path('feed/', LatestPostsFeed(), name='post_feed'),
]
```

در مرورگر خود بروید. اکنون باید فید RSS ، از جمله پنج پست آخر وبلاگ را مشاهده کنید:

```
<?xml version="1.0" encoding="utf-8"?>
<rss xmlns:atom="http://www.w3.org/2005/Atom" version="2.0">
  <channel>
    <title>My blog</title>
    <link>http://localhost:8000/blog/</link>
    <description>New posts of my blog.</description>
    <atom:link href="http://localhost:8000/blog/feed/" rel="self"/>
    <language>en-us</language>
    <lastBuildDate>Fri, 15 Dec 2017 09:56:40 +0000</lastBuildDate>
    <item>
      <title>Who was Django Reinhardt?</title>
      <link>http://localhost:8000/blog/2017/12/14/who-was-django-reinhardt/</link>
      <description>Who was Django Reinhardt.</description>
      <guid>http://localhost:8000/blog/2017/12/14/who-was-django-reinhardt/</guid>
    </item>
    ...
  </channel>
</rss>
```

اگر همان URL را در یک سرویس گیرنده RSS باز کنید ، می توانید فید خود را با یک رابط کاربر پسند مشاهده کنید.

مرحله آخر اضافه کردن لینک اشتراک به نوار کناری وبلاگ است.

الگوی blog / base.html را باز کنید و زیر تعداد کل پست های داخل بخش نوار کناری خط زیر را اضافه کنید:

```
<p><a href="{% url "blog:post_feed" %}">Subscribe to my RSS feed</a></p>
```

اکنون در مرورگر خود `/blog` را باز کنید و به نوار کناری نگاهی بیندازید. پیوند جدید باید شما را به فید وبلاگ شما برد:

My blog

This is my blog. I've written 5 posts so far.

[Subscribe to my RSS feed](#)

جستجوی متن کامل را به وبلاگ خود اضافه کنید

اکنون ، قابلیت های جستجو را به وبلاگ خود اضافه خواهید کرد. Django ORM به شما امکان می دهد تا با استفاده از مثال هایی که حاوی یک فیلتر (یا نسخه غیر حساس آن است) ، عملیات تطبیق ساده را انجام دهید. می توانید از جستجوی زیر برای یافتن پست هایی که حاوی کلمه چارچوب در بدن آنها است استفاده کنید:

```
from blog.models import Post  
Post.objects.filter(body__contains='framework')
```

با این حال ، اگر می خواهید جستجوی پیچیده جستجو را انجام دهید ، نتایج را با مشابهت یا با اصطلاح های وزنی بازیابی کنید ، باید از موتور جستجوی متن کامل استفاده کنید.

Django قابلیت جستجوی قدرتمندی را در بالای ویژگی های جستجوی متن کامل PostgreSQL ایجاد می کند. مازوں django.contrib.Postgres عملکردهایی را ارائه می دهد که توسط PostgreSQL ارائه شده است و در پایگاه داده های Django که از آنها پشتیبانی می کند ، به اشتراک گذاشته نمی شود. می توانید در مورد جستجوی متن کامل دیگری که از آنها بیاموزید. <https://www.postgresql.org/docs/10/static/textsearch.html> در PostgreSQL

نصب PostgreSQL

شما در حال حاضر از SQLite برای پروژه و بلاگ خود استفاده می کنید. این برای اهداف توسعه کافی است. با این حال ، برای یک محیط تولید ، به یک پایگاه داده قدرتمندتر ، مانند Oracle ، MySQL یا PostgreSQL نیاز دارید. ما پایگاه داده خود را به PostgreSQL تغییر می دهیم تا از ویژگی های جستجوی متن کامل آن بهره بیریم. اگر از لینوکس استفاده می کنید ، وابستگی هایی را برای PostgreSQL برای همکاری با پایتون نصب کنید ، مانند این:

```
sudo apt-get install libpq-dev python-dev
```

سپس PostgreSQL را با دستور زیر نصب کنید:

```
sudo apt-get install postgresql postgresql-contrib
```

اگر از Windows یا MacOS X استفاده می کنید ، PostgreSQL را از <https://www.postgresql.org/download> را از بارگیری کنید و نصب کنید.

همچنین باید آداتور PostgreSQL را برای Python نصب کنید. برای نصب آن ، دستور زیر را در پوسته اجرا کنید:

```
pip install psycopg2==2.7.4
```

باید برای بانک اطلاعاتی PostgreSQL مارکر ایجاد کنیم. پوسته را باز کنید و دستورات زیر را اجرا کنید:

```
su postgres  
createuser -dP blog
```

برای کاربر جدید از شما رمزعبور خواسته می شود. رمزعبور مورد نظر خود را وارد کرده و سپس بانک اطلاعات وبلاگ را ایجاد کرده و مالکیت آنرا به کاربر وبلاگ که اخیراً با دستور زیر ایجاد کرده اید ، واگذار کنید:

```
createdb -E utf8 -U blog blog
```

سپس پرونده setting.py پروژه خود را ویرایش کرده و تنظیمات DATABASES را اصلاح کنید تا به صورت زیر ظاهر شود:

```
DATABASES = {  
    'default': {  
        'ENGINE': 'django.db.backends.postgresql',  
        'NAME': 'blog',  
        'USER': 'blog',  
        'PASSWORD': '*****',  
    }  
}
```

داده های قبلی را با نام بانک اطلاعاتی و اعتبارنامه های مربوط به کاربر مورد نظر خود جایگزین کنید. بانک اطلاعاتی جدید خالی است. دستور زیر را اجرا کنید تا کلیه مهاجرت های پایگاه داده اعمال شود:

```
python manage.py migrate
```

سرانجام با دستور زیر یک superuser ایجاد کنید:

```
python manage.py createsuperuser
```

هم اکنون می توانید سرور توسعه را اجرا کرده و به سرپرست سایت به آدرس <http://127.0.0.1:8000/admin> با admin جدید دسترسی پیدا کنید.

از آنجاکه ما پایگاه داده را تغییر دادیم ، هیچ پستی در آن ذخیره نمی شود. پایگاه داده جدید خود را با چند نمونه از وبلاگ های نمونه جمع کنید تا بتوانید جستجوها را در برابر پایگاه داده انجام دهید.

جستجوی ساده

پرونده settings.py اضافه INSTALLED_APPS را به تنظیمات django.contrib.postgres و ویرایش کنید و پروژه خود را ویرایش کنید ، به شرح زیر:

```
INSTALLED_APPS = [  
    # ...  
    'django.contrib.postgres',  
]
```

اکنون می توانید با استفاده از جستجوی QuerySet ، یک زمینه واحد را جستجو کنید ، مانند این:

```
from blog.models import Post  
Post.objects.filter(body__search='django')
```

این پرس و جو از PostgreSQL برای ایجاد یک وکتور جستجو برای قسمت body و یک جستجو از اصطلاح جنگو استفاده می کند. نتایج با مطابقت پرس و جو با بردار به دست می آیند.

جستجو در مقابل چندین زمینه

ممکن است بخواهید در قسمتهای مختلف جستجو کنید. در این حالت ، شما باید SearchVector را تعریف کنید. بایاید یک بردار بسازیم که به ما امکان جستجو در برابر عنوان و قسمتهای بدنی مدل Post را بدهد:

```
from django.contrib.postgres.search import SearchVector
from blog.models import Post

Post.objects.annotate(
    search=SearchVector('title', 'body'),
).filter(search='django')
```

با استفاده از حاشیه نویسی و تعریف SearchVector با هر دو زمینه ، ما عملکردی را برای مطابقت با پرس و جو در برابر عنوان و بدنی پست ها ارائه می دهیم.

ایجاد نمای جستجو

اکنون ، ما یک نمای سفارشی ایجاد خواهیم کرد تا به کاربرانمان امکان جستجوی پست ها را بدهند. ابتدا به فرم جستجو نیاز خواهیم داشت. فایل forms.py برنامه وبلاگ را ویرایش کنید و فرم زیر را اضافه کنید:

```
class SearchForm(forms.Form):
    query = forms.CharField()
```

ما از قسمت query استفاده می کنیم تا به کاربران اجازه دهیم اصطلاحات جستجو را معرفی کنند. پرونده views.py برنامه وبلاگ را ویرایش کنید و کد زیر را به آن اضافه کنید:

```
from django.contrib.postgres.search import SearchVector
from .forms import EmailPostForm, CommentForm, SearchForm

def post_search(request):
    form = SearchForm()
    query = None
    results = []
    if 'query' in request.GET:
        form = SearchForm(request.GET)
        if form.is_valid():
            query = form.cleaned_data['query']
            results = Post.objects.annotate(
                search=SearchVector('title', 'body'),
            ).filter(search=query)
    return render(request,
                  'blog/post/search.html',
                  {'form': form,
                   'query': query,
                   'results': results})
```

در نمای قبلی ابتدا فرم SearchForm را فوری می کنیم. ما قصد داریم تا فرم را با استفاده از روش GET ارسال کنیم تا حاصل از آن پارامتر پرس و جو را شامل شود. برای بررسی اینکه فرم ارائه شده است ، ما به دنبال پارامتر پرس و جو در فرهنگ لغت درخواست هستیم. GET. هنگامی که فرم ارسال شد ، ما آن را با داده های GET ارسال شده فوراً ارسال می کنیم و تأیید می کنیم که داده فرم معتبر است. اگر فرم معتبر است ، ما پستهایی را با نمونه SearchVector سفارشی که با عنوان و قسمت های بدن ساخته شده است جستجو می کنیم.

اکنون نمای جستجو آماده است. برای نمایش فرم و نتایج هنگام استفاده کاربر ، باید یک الگوی ایجاد کنیم. یک پرونده جدید در فهرست / blog / post / الگو ایجاد کنید ، نام آن را جستجو کنید.html و کد زیر را به آن اضافه کنید:

```

{% extends "blog/base.html" %}

{% block title %}Search{% endblock %}

{% block content %}
{% if query %}
<h1>Posts containing "{{ query }}"</h1>
<h3>
    {% with results.count as total_results %}
        Found {{ total_results }} result{{ total_results|pluralize }}
    {% endwith %}
</h3>
{% for post in results %}
    <h4><a href="{{ post.get_absolute_url }}">{{ post.title }}</a></h4>
    {{ post.body|truncatewords:5 }}
{% empty %}
    <p>There are no results for your query.</p>
{% endfor %}
<p><a href="{% url "blog:post_search" %}">Search again</a></p>
{% else %}
<h1>Search for posts</h1>
<form action"." method="get">
    {{ form.as_p }}
    <input type="submit" value="Search">
</form>
{% endif %}
{% endblock %}

```

همانطور که در نمای جستجوی توانیم تشخیص دهیم که فرم با حضور پارامتر p_{rs} و جو ارسال شده است یا خیر. قبل از ارسال پست، فرم و دکمه ارسال را نمایش می دهیم. پس از ارسال پست، پرس و جو انجام شده، تعداد کل نتایج و لیست ارسال های برگشت داده شده را نمایش می دهیم.

در آخر، پرونده urls.py برنامه وبلاگ خود را ویرایش کنید و الگوی URL زیر را اضافه کنید:

```
path('search/', views.post_search, name='post_search'),
```

اکنون در مرورگر خود <http://127.0.0.1:8000/blog/search> را باز کنید. باید فرم جستجو زیر را مشاهده کنید:

Search for posts

Query:

SEARCH

یک پرس و جو وارد کنید و بر روی دکمه جستجو کلیک کنید. نتایج جستجوی جستجو را به شرح زیر مشاهده خواهید کرد:

Posts containing "music"

Found 2 results

[Another post more](#)

Post body.

[Who was Django Reinhardt?](#)

The Django web framework was ...

[Search again](#)

My blog

This is my blog. I've written 4 posts so far.

[Subscribe to my RSS feed](#)

Latest posts

- [Another post more](#)
- [New title](#)
- [Who was Django Reinhardt?](#)

Most commented posts

- [Who was Django Reinhardt?](#)
- [New title](#)
- [Another post more](#)
- [Old](#)

تبریک می‌گوییم! شما یک موتور جستجوی اساسی برای وبلاگ خود ایجاد کرده‌اید.

نتایج تقویت شده و رتبه بندی

یک کلاس `SearchQuery` برای ترجمه اصطلاحات در یک موضوع جستجو فراهم می‌کند. به طور پیش فرض، شرایط از طریق الگوریتم‌های مبدل منتقل می‌شود، که به شما کمک می‌کند مسابقات بهتری را بدست آورید.

همچنین ممکن است بخواهید نتایج را با توجه به سفارش سفارشی کنید. `PostgreSQL` یک عملکرد رتبه بندی را فراهم می‌کند که نتایج را بر اساس تعداد بازدید از اصطلاحات نمایش داده شده و نزدیک بودن آنها، سفارش می‌دهد. پرونده `views.py` برنامه وبلاگ خود را ویرایش کنید و واردات زیر را اضافه کنید:

```
from django.contrib.postgres.search import SearchVector, SearchQuery,  
SearchRank
```

سپس به خطوط زیر نگاهی بیندازید:

```
results = Post.objects.annotate(  
    search=SearchVector('title', 'body'),  
) .filter(search=query)
```

آنها را با موارد زیر جایگزین کنید:

```
search_vector = SearchVector('title', 'body')  
search_query = SearchQuery(query)  
results = Post.objects.annotate(  
    search=search_vector,  
    rank=SearchRank(search_vector, search_query)  
) .filter(search=search_query).order_by('-rank')
```

در کد قبلی ، یک شی SearchQuery ایجاد کردیم ، نتایج توسط آن فیلتر شده و از SearchRank برای سفارش دادن نتایج با توجه استفاده می کنیم. می توانید `/http://127.0.0.1:8000/blog/search/` را در مرورگر خود باز کنید و جستجوهای مختلف را برای تست stemment و رتبه بندی آزمایش کنید. در زیر نمونه ای از رتبه بندی بر اساس تعداد وقایع مربوط به کلمه django در عنوان و بدنه پست ها است:

Posts containing "django"

Found 3 results

[Django, Django, Django](#)

Django is the Web Framework ...

[Django twice](#)

Django offers full text search ...

[Django once](#)

A Python web framework.

[Search again](#)

Weighting queries

شما می توانید بردارهای خاص را تقویت کنید به طوری که هنگام سفارش نتایج با توجه به وزن بیشتری به آنها منتقل می شود. به عنوان مثال ، می توانید از این طریق استفاده کنید تا به پستهایی که با عنوان مطابقت دارند بیشتر توجه کنید تا محتوای خطوط قبلي پرونده `views.py` برنامه وبلاگ خود را ویرایش کنید و آنها را به این شکل جلوه دهید:

```
search_vector = SearchVector('title', weight='A') + SearchVector('body', weight='B')
search_query = SearchQuery(query)
results = Post.objects.annotate(
    rank=SearchRank(search_vector, search_query)
).filter(rank__gte=0.3).order_by('-rank')
```

در کد قبلی ، وزنهای مختلف را در بردارهای جستجو ایجاد شده با استفاده از عنوان و فیلدهای بدن اعمال می کنیم. وزنهای پیش فرض D ، C ، B و A هستند که به ترتیب به شماره های 0.1 ، 0.2 ، 0.4 و 1.0 اشاره می کنند.

ما یک وزن 1.0 در بردار جستجوی عنوان و وزن 0.4 به بردار بدن اعمال می کنیم: عنوان بر محتوای بدن غلبه می کند. ما نتایج را فیلتر می کنیم تا فقط موارد با رتبه بالاتر از 0.3 نمایش داده شود.

جستجو با شباهت trigram

روش جستجوی دیگر ، شباهت trigram است. تریگرام گروهی از سه شخصیت متواالی است. شما می توانید شباهت تعداد دو رشته را با شمارش تعداد تراریگ های اشتراک شده آنها اندازه گیری کنید. به نظر می رسد این روش برای اندازه گیری شباهت کلمات در بسیاری از زیانها بسیار مؤثر است.

برای استفاده از تراریگ ها در PostgreSQL ، ابتدا باید پسوند pg_trgm را نصب کنید. برای اتصال به پایگاه داده خود دستور زیر را از پوسته اجرا کنید:

```
psql blog
```

سپس دستور زیر را برای نصب پسوند pg_trgm اجرا کنید:

```
CREATE EXTENSION pg_trgm;
```

بیایید نمای خود را ویرایش کنیم و آن را اصلاح کنیم تا trigrams را جستجو کنیم. پرونده views.py برنامه وبلاگ خود را ویرایش کنید و واردات زیر را اضافه کنید:

```
from django.contrib.postgres.search import TrigramSimilarity
```

سپس عبارت جستجوی ارسال را با خطوط زیر جایگزین کنید:

```
results = Post.objects.annotate(
    similarity=TrigramSimilarity('title', query),
).filter(similarity__gt=0.3).order_by('-similarity')
```

/ra در مرورگر خود باز کنید و جستجوهای مختلفی را برای تراویگام ها آزمایش کنید. مثلا زیر یک تایپ فرضی را در اصطلاح django نشان می دهد:

Posts containing "yango"

Found 1 result

[Django Django](#)

A Python web framework.

اکنون ، شما یک موتور جستجوی قدرتمند در پروژه خود دارید. می توانید اطلاعات بیشتر در مورد جستجوی متن کامل را در <https://docs.djangoproject.com/en/2.0/ref/contrib/postgres/search> پیدا کنید.

سایر موتورهای جستجو متن کامل

ممکن است بخواهید از موتور جستجوی متن کامل با PostgreSQL استفاده کنید. اگر می خواهید از Solr یا Elasticsearch استفاده کنید ، می توانید با استفاده از آنها را در پروژه Haystack خود ادغام کنید. یک برنامه Django است که به عنوان یک لایه انتزاع برای موتورهای جستجوگر متعدد کار می کند. این یک API جستجو ساده بسیار شبیه به Django QuerySets ارائه می دهد. می توانید اطلاعات بیشتر در مورد Haystack را در <http://haystacksearch.org> پیدا کنید.

خلاصه

در این فصل ، شما یاد گرفتید که چگونه برچسب ها و فیلترهای قالب دلخواه سفارشی را تهیه کنید تا قالب ها از قابلیت های سفارشی برخوردار شوند.

شما همچنین یک نقشه سایت برای موتورهای جستجو برای خزیدن سایت خود و یک فید RSS برای کاربران برای عضویت در وبلاگ خود ایجاد کرده اید. شما همچنین با استفاده از موتور جستجوی متن کامل PostgreSQL ، یک موتور جستجو برای وبلاگ خود ایجاد کرده اید. در فصل بعد ، شما می آموزید که چگونه با استفاده از چارچوب تأیید هویت ، Django ایجاد یک وب سایت اجتماعی ، ایجاد پروفایل کاربر سفارشی و ساخت احراز هویت اجتماعی ، ایجاد کنید.

فصل چهارم

ساختن یک

وب سایت

شبکه اجتماعی

در فصل قبل ، شما یاد گرفتید که چگونه نقشه های مکانی و فید ها را ایجاد کنید و یک موتور جستجو برای برنامه وبلاگ خود ایجاد کرده اید. در این فصل یک برنامه کاربردی اجتماعی تهیه خواهید کرد. شما می توانید عملکردی را برای ورود به سیستم ، خروج از سیستم ، ویرایش و تنظیم مجدد رمز ورود خود ایجاد کنید. شما یاد می گیرید که چگونه یک پروفایل شخصی برای کاربران خود ایجاد کنید ، و احراز هویت اجتماعی را به سایت خود اضافه کنید. این فصل موضوعات زیر را شامل می شود:

با استفاده از چارچوب احراز هویت Django

ایجاد نماهای ثبت نام کاربر

گسترش مدل کاربر با یک مدل پروفایل سفارشی

اضافه کردن احراز هویت اجتماعی با پایتون-اجتماعی-نویسنده

بایاید با ایجاد پروژه جدید خود شروع کنیم.

ایجاد یک پروژه وب سایت اجتماعی

ما یک برنامه اجتماعی ایجاد خواهیم کرد که به کاربران امکان می دهد تا تصاویر خود را در اینترنت به اشتراک بگذارند. برای این پروژه باید عناصر زیر را بسازیم:

یک سیستم تأیید اعتبار برای کاربران برای ثبت نام ، ورود به سیستم ، ویرایش مشخصات خود و تغییر یا تنظیم مجدد رمز ورود خود

یک سیستم دنبال کننده برای این که کاربران بتوانند از یکدیگر پیروی کنند عملکردی برای نمایش تصاویر مشترک و پیاده سازی یک نشانک برای کاربران برای به اشتراک گذاشتن تصاویر از هر وب سایت

یک جریان فعالیت برای هر کاربر که به کاربران امکان می دهد محتوای آپلود شده توسط افرادی که دنبال می کنند را ببینند. این فصل به اولین نکته ذکر شده در لیست قبلی می پردازد.

شروع پروژه وب سایت اجتماعی خود

ترمینال را باز کنید و از دستورات زیر استفاده کنید تا یک محیط مجازی برای پروژه خود ایجاد کنید و آن را فعال کنید:

```
mkdir env  
virtualenv env/bookmarks  
source env/bookmarks/bin/activate
```

اعلان پوسته محیط مجازی فعال شما را به شرح زیر نشان می دهد:

```
(bookmarks)laptop:~ zenx$
```

را در محیط مجازی خود با دستور زیر نصب کنید:

```
pip install Django
```

دستور زیر را برای ایجاد یک پروژه جدید اجرا کنید:

```
django-admin startproject bookmarks
```

پس از ایجاد ساختار اولیه پروژه ، از دستورات زیر استفاده کنید تا وارد فهرست پروژه خود شوید و یک برنامه جدید با نام account ایجاد کنید:

```
cd bookmarks/  
django-admin startapp account
```

به یاد داشته باشید که باید برنامه خود را در پروژه خود با افزودن آن به تنظیمات INSTALLED_APPS در پرونده setting.py فعال کنید.

قبل از هر برنامه نصب شده دیگر ، آن را در لیست INSTALLED_APPS قرار دهید:

```
INSTALLED_APPS = [  
    'account.apps.AccountConfig',  
    # ...  
]
```

بعداً الگوهای تأیید صحت جنگو را تعریف خواهیم کرد. با قرار دادن برنامه برای اولین بار در تنظیمات INSTALLED_APPS ، اطمینان حاصل می کنیم که الگوهای تأیید اعتبار ما بجای سایر الگوهای احراز هویت موجود در سایر برنامه ها به طور پیش فرض استفاده می شوند. جنگو به ترتیب ظاهر برنامه در تنظیمات INSTALLED_APPS به دنبال الگوها میگردد.

دستور بعدی را برای همگام سازی بانک اطلاعاتی با مدل های برنامه های پیش فرض موجود در تنظیمات INSTALLED_APPS اجرا کنید:

```
python manage.py migrate
```

خواهید دید که کلیه مراحل اولیه انتقال پایگاه داده Django اعمال می شود.

ما با استفاده از چارچوب احراز هویت Django ، یک سیستم تأیید اعتبار را در پروژه خود ایجاد خواهیم کرد.

استفاده از چارچوب احراز هویت Django

Django دارای یک چارچوب تأیید هویت داخلی است که می تواند احراز هویت کاربر ، جلسات ، مجوزها و گروه های کاربر را کنترل کند. سیستم احراز هویت شامل نمایی برای اقدامات مشترک کاربر مانند ورود به سیستم ، تغییر رمز عبور و تنظیم مجدد رمز عبور است. چارچوب احراز هویت در django.contrib.auth واقع شده است و توسط سایر بسته های کمک کننده Django استفاده می شود. به یاد داشته باشید که قبل از چارچوب تأیید اعتبار در فصل 1 ، ساختن یک برنامه وبلاگ استفاده کرده اید ، تا یک برنامه سرگرمی برای برنامه وبلاگ خود ایجاد کنید تا به سایت مدیریت دسترسی پیدا کند.

وقتی با استفاده از دستور `start` یک پروژه جدید Django ایجاد می کنید ، چارچوب تأیید اعتبار در تنظیمات پیش فرض پروژه شما گنجانده می شود. این برنامه شامل برنامه `django.contrib.auth` و دو کلاس میانی زیر است که در تنظیمات MiddleWARE پروژه شما یافت شده است:

کاربران را با درخواست ها با استفاده از جلسات مرتبط می کند: AuthenticationMiddleware

جلسه فعلی را با درخواستها انجام می دهد: SessionMiddleware

کلاس با روشهای است که در مرحله درخواست یا پاسخ به صورت جهانی اجرا می شود. شما در سراسر این Middleware کتاب چندین بار از کلاس های میانی استفاده خواهید کرد ، و می آموزید که وسایل واسطه های سفارشی را در فصل 13 ، Going Live ایجاد کنید.

چارچوب احراز هویت همچنین مدل های زیر را شامل می شود:

کاربر: یک مدل کاربر با فیلد های اساسی. زمینه های اصلی این مدل عبارتند از نام کاربری ، رمز عبور ، ایمیل ، first_name و last_name is_active

گروه: یک مدل گروهی برای طبقه بندی کاربران

مجوز: پرچم هایی برای کاربران یا گروه ها برای انجام برخی اقدامات خاص

این چارچوب همچنین شامل نماهای پیش فرض تأیید اعتبار و اشکال است که بعداً از آنها استفاده خواهیم کرد.

Creating a login view

ما این بخش را با استفاده از چارچوب تأیید هویت Django شروع می کنیم تا کاربران بتوانند به وب سایت ما وارد شوند. برای ورود به سیستم کاربر ، نمایه ما باید اقدامات زیر را انجام دهد:

1. با ارسال فرم ، نام کاربری و رمز عبور را بدست آورید

2- کاربر را در برابر داده های ذخیره شده در پایگاه داده تأیید کنید

3. بررسی کنید که کاربر فعال است یا خیر

4- کاربر را وارد وب سایت کنید و یک جلسه تأیید شده را شروع کنید

ابتدا فرم ورود به سیستم ایجاد خواهیم کرد. یک پرونده جدید `form.py` را در فهرست برنامه کاربردی حساب خود ایجاد کنید و خطوط زیر را به آن اضافه کنید:

```
from django import forms

class LoginForm(forms.Form):
    username = forms.CharField()
    password = forms.CharField(widget=forms.PasswordInput)
```

از این فرم برای تأیید اعتبار کاربران در برابر دیتابیس استفاده می شود.

توجه داشته باشید که ما از ویجت `PasswordInput` برای ارائه عنصر ورودی HTML آن ، از جمله ویژگی "استفاده می کنیم ، به گونه ای که مرورگر از آن به عنوان ورودی رمز عبور استفاده می کند. پرونده `views.py` برنامه حساب خود را ویرایش کنید و کد زیر را به آن اضافه کنید:

```
from django.http import HttpResponseRedirect
from django.shortcuts import render
from django.contrib.auth import authenticate, login
from .forms import LoginForm
```

```

def user_login(request):
    if request.method == 'POST':
        form = LoginForm(request.POST)
        if form.is_valid():
            cd = form.cleaned_data
            user = authenticate(request,
                                username=cd['username'],
                                password=cd['password'])
            if user is not None:
                if user.is_active:
                    login(request, user)
                    return HttpResponseRedirect('Authenticated \'\\
                                              \'successfully')
                else:
                    return HttpResponseRedirect('Disabled account')
            else:
                return HttpResponseRedirect('Invalid login')
        else:
            form = LoginForm()
    return render(request, 'account/login.html', {'form': form})

```

این همان کاری است که نمای ورود به سیستم اصلی ما انجام می دهد: وقتی نمای user_login با یک درخواست GET فراخوانی می شود ، یک فرم ورود جدید را با () form = LoginForm(). برای نمایش آن در قالب آنی فوری می کنیم. وقتی کاربر فرم را از طریق POST ارسال می کند ، اقدامات زیر را انجام می دهیم:

1. فرم را با داده های ارسال شده (request.POST.form = LoginForm()) فوری کنید.
2. بررسی کنید که آیا فرم با form.is_valid معتبر است یا خیر. اگر صحت ندارد ، ملاحظاتی فرم را در الگوی خود نمایش می دهیم (برای مثال ، اگر کاربر یکی از فیلدها را پر نکرد).
3. اگر داده های ارسالی معتبر هستند ، ما با استفاده از روش تأیید اعتبار ، کاربر را در برابر دیتابیس تأیید می کنیم. در این روش شیء درخواست ، نام کاربری و پارامترهای رمزعبور به کار می رود و اگر کاربر با موقفيت تأیید شده باشد ، کاربر مورد نظر را برمی گرداند یا در غیر اين صورت هیچ یك از آنها را انجام نمی دهد. اگر کاربر تأیید نشده است ، ما یک HttpResponseRedirect را به شما باز می گردانیم ، پیام ورود نامعتبر را نشان می دهیم.

4- اگر کاربر با موفقیت تأیید شد ، بررسی می کنیم که کاربر فعال است ، آیا به ویژگی is_active خود دسترسی پیدا می کند. این یک ویژگی از مدل کاربر جنگو است. اگر کاربر فعال نیست ، ما HttpResponse را که پیام حساب غیرفعال شده را نشان می دهد ، باز می گردیم.

5- در صورت فعال بودن کاربر ، کاربر را وارد وب سایت می کنیم. ما با فراخوانی متده login (کاربر) کاربر را در جلسه تنظیم می کنیم و پیام تأیید شده را با موفقیت باز می گردیم.

اکنون ، برای ایجاد این الگوی باید الگوی URL ایجاد کنید. یک پرونده urls.py جدید را در فهرست برنامه کاربردی حساب خود ایجاد کنید و کد زیر را به آن اضافه کنید:

```
from django.urls import path
from . import views
urlpatterns = [
    # post views
    path('login/', views.user_login, name='login'),
```

پرونده اصلی urls.py واقع در فهرست پروژه های نشانک خود را ویرایش کنید ، واردات را وارد کنید و الگوهای URL برنامه حساب را به شرح زیر اضافه کنید:

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

urlpatterns = [
    path('admin/', admin.site.urls),
    path('account/', include('account.urls')),
```

اکنون از طریق URL می توانید به نمای ورود دسترسی پیدا کنید. زمان آن رسیده است که یک الگوی برای این نمای ایجاد کنید. از آنجا که هیچ الگوی برای این پروژه ندارید ، می توانید با ایجاد یک الگوی پایه که می تواند توسط الگوی ورود به سیستم گسترش یابد ، شروع کنید. پرونده ها و فهرستهای زیر را در فهرست برنامه کاربردی حساب ایجاد کنید:

```
templates/  
  account/  
    login.html  
  base.html
```

پرونده را ویرایش کنید و کد زیر را به آن اضافه کنید:

```
{% load staticfiles %}  
<!DOCTYPE html>  
<html>  
<head>  
  <title>{% block title %}{% endblock %}</title>  
  <link href="{% static "css/base.css" %}" rel="stylesheet">  
</head>  
<body>  
  <div id="header">  
    <span class="logo">Bookmarks</span>  
  </div>  
  <div id="content">  
    {% block content %}  
    {% endblock %}  
  </div>  
</body>  
</html>
```

این الگوی اصلی وب سایت خواهد بود.

همانطور که در پروژه قبلی خود انجام دادیم ، سبک های CSS را در قالب اصلی قرار می دهیم.

شما می توانید این پرونده های استاتیک را در کدی که همراه با این فصل است پیدا کنید. استاتیک / دایرکتوری برنامه حساب را از کد منبع فصل در همان مکان در پروژه خود کپی کنید تا بتوانید از فایلهای استاتیک استفاده کنید.

قالب پایه یک بلوک عنوان و یک بلوک محتوا را تعریف می کند که توسط قالب هایی که از آن گسترش می یابند می توانند با محتوا پر شوند.

باید قالب مربوط به فرم ورود خود را پر کنیم. Account/login.html را باز کنید.

قالب را اضافه کنید و کد زیر را به آن اضافه کنید:

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

{% block title %}Log-in{% endblock %}

{% block content %}
<h1>Log-in</h1>
<p>Please, use the following form to log-in:</p>
<form action"." method="post">
{{ form.as_p }}
{% csrf_token %}
<p><input type="submit" value="Log in"></p>
</form>
{% endblock %}
```

این الگو شامل شکلی است که در نمای مشاهده می شود.

از آنجاکه فرم ما از طریق POST ارسال می شود، {٪}%

قالب {٪}% csrf_token برای حفاظت از CSRF. شما در مورد حفاظت از CSRF در فصل 2 ، تقویت و بلاگ خود با ویژگی های پیشرفته آموخته اید.

هنوز هیچ کاربر در پایگاه داده شما وجود ندارد. برای اینکه بتوانید به سایت مدیریت دسترسی پیدا کنید برای مدیریت سایر کاربران ، ابتدا باید یک superuser ایجاد کنید. خط فرمان را باز کرده و python management.py را ایجاد کنید اجرا کنید. نام کاربری ، ایمیل و رمز عبور مورد نظر خود را وارد کنید. سپس سرور توسعه را با استفاده از python management.py اجرا کنید

دستور runserver را بزنید و <http://127.0.0.1:8000/admin> را در مرورگر خود باز کنید.

با استفاده از اعتبار کاربری که اخیراً ایجاد کرده اید به سایت مدیریت دسترسی پیدا کنید. شما می توانید سایت دولت Django را مشاهده کنید ، از جمله مدل های کاربر و گروه چارچوب تأیید هویت Django. به شرح زیر خواهد بود:



با استفاده از سایت مدیریت و یک کاربر جدید ایجاد کنید <http://127.0.0.1:8000/account/login> در مرورگر خود باز کنید. شما باید الگوهای ارائه شده ، از جمله فرم ورود را ببینید:



Log-in

Please, use the following form to log-in:

Username:

Password:

LOG IN

اکنون فرم را ارسال کنید و یکی از قسمت ها را خالی بگذارید. در این حالت ، خواهید دید که فرم معتبر نیست و خطاهای را به شرح زیر نشان می دهد:

Username:

test

This field is required.

Password:

LOG IN

توجه داشته باشید که برخی مروگرهای مدرن از ارسال فرم با قسمتهای خالی یا اشتباه جلوگیری می‌کنند. این امر به دلیل اعتبارسنجی فرم است که توسط مروگر مبتنی بر انواع فیلدها و محدودیت‌ها در هر زمینه انجام می‌شود. در این حالت، فرم ارسال نمی‌شود و مروگر پیغام خطایی را برای قسمت‌های اشتباه نشان می‌دهد.

اگر یک کاربر غیر موجود یا یک رمز عبور اشتباه وارد کنید، یک پیام ورود نامعتبر دریافت خواهد کرد.

اگر اعتبارنامه معتبری را وارد کنید، یک پیام معتبر با موفقیت دریافت خواهد کرد، مانند این:



Authenticated successfully

استفاده از نماهای تأیید هویت Django

جنگو شامل چندین شکل و دیدگاه در چارچوب تأیید اعتبار است که می‌توانید بلافضله از آنها استفاده کنید. نمای ورود به سیستم که ایجاد کرده اید یک تمرين خوب برای درک روند احراز هویت کاربر در جنگو است. با این وجود، در بیشتر موارد می‌توانید از نماهای پیش فرض تأیید هویت Django استفاده کنید.

جنگو برای مقابله با تأیید اعتبار، دیدگاههای مبتنی بر کلاس را در زیر ارائه می‌دهد. همه آنها در django.contrib.auth.views قرار دارند:

LoginForm: فرم ورود به سیستم را کنترل می‌کند و در یک کاربر وارد می‌شود

LogoutView: کاربر را از سیستم خارج می‌کند

Django نمایش‌های زیر را برای رسیدگی به تغییرات رمز عبور ارائه می‌دهد:

PasswordChangeView: یک فرم برای تغییر رمز عبور کاربر اعمال می‌شود

PasswordChangeDoneView: نمای موفقیت پس از تغییر رمز عبور موفقیت آمیز به کاربر هدایت می‌شود

همچنین Django شامل نمایش‌های زیر می‌باشد تا به کاربران امکان تنظیم مجدد رمز ورود خود را بدهد:

PasswordResetView: به کاربران امکان می‌دهد رمز عبور خود را دوباره تنظیم کنند.

من یک لینک استفاده یکبار را با یک نشان تولید می‌کنم و آن را به حساب ایمیل کاربر ارسال می‌کنم.

PasswordResetDoneView: به کاربران می‌گوید که ایمیلی از جمله پیوند برای تنظیم مجدد گذرواژه خود برای آنها ارسال شده است.

PasswordResetConfirmView: به کاربران امکان می‌دهد رمز عبور جدیدی تنظیم کنند.

نمای موفقیت پس از تنظیم مجدد رمز عبور ، به کاربر هدایت می شود: PasswordResetCompleteView

نمایش های ذکر شده در لیست قبلی می توانند در هنگام ایجاد وب سایت با حساب های کاربری ، زمان زیادی را برای شما ذخیره کنند. نماها از مقادیر پیش فرض استفاده می کنند که می توانید آنها را نادیده بگیرید ، مانند مکان قالب برای ارائه یا فرم مورد استفاده برای نمایش.

می توانید اطلاعات بیشتر در مورد نماهای تأیید اعتبار داخلی را در دریافت <https://docs.djangoproject.com/en/2.0/topics/auth/default/#all-authentication-views> کنید.

Login and logout views

برنامه حساب خود را مانند این ویرایش کنید: urls.py

```
from django.urls import path
from django.contrib.auth import views as auth_views
from . import views

urlpatterns = [
    # previous login view
    # path('login/', views.user_login, name='login'),
    path('login/', auth_views.LoginView.as_view(), name='login'),
    path('logout/', auth_views.LogoutView.as_view(), name='logout'),
]
```

ما از الگوی URL برای نمایه user_login که قبلاً ایجاد کرده ایم برای استفاده از نمای LoginView از چارچوب احراز هویت Django اظهار نظر می کنیم. ما همچنین یک الگوی URL برای نمای LogoutView اضافه می کنیم.

یک فهرست جدید در فهرست برنامه های حساب خود ایجاد کنید و نام آن را ثبت کنید. این مسیری پیش فرض است که در آن بازدیدهای تأیید هویت Django انتظار دارند که الگوهای تأیید اعتبار شما در آن باشد.

ماژول django.contrib.admin شامل برخی از الگوهای تأیید اعتبار است که برای سایت مدیریت استفاده می شود. ما برنامه حساب را در بالای تنظیمات INSTALLED_APPS قرار داده ایم به گونه ای که جنگو بجای هر قالب تأیید اعتبار تعریف شده در برنامه های دیگر، به طور پیش فرض از الگوهای ما استفاده می کند.

یک پرونده جدید در داخل templates/registration ایجاد کنید، نام آن را login.html نامگذاری کنید و کد زیر را به آن اضافه کنید:

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

{% block title %}Log-in{% endblock %}

{% block content %}
<h1>Log-in</h1>
{% if form.errors %}


Your username and password didn't match.  

    Please try again.


{% else %}


Please, use the following form to log-in:


{% endif %}
<div class="login-form">
  <form action="{% url 'login' %}" method="post">
    {{ form.as_p }}
    {% csrf_token %}
    <input type="hidden" name="next" value="{{ next }}" />
    <p><input type="submit" value="Log-in"></p>
  </form>
</div>
{% endblock %}
```

این الگوی ورود کاملاً شبیه به الگویی است که قبل ایجاد کردیم.

جنگو بطور پیش فرض از فرم AuthenticationForm واقع در django.contrib.auth.forms استفاده می کند. این فرم سعی در تأیید اعتبار کاربر دارد و در صورت عدم موفقیت ورود به سیستم خطای اعتبار سنجی را ایجاد می کند. در این حالت ، می توانیم با استفاده از % if form.errors در قالب خطا هستیم ، بررسی کنیم که اعتبار ارائه شده اشتباه است یا خیر. توجه داشته باشید که ما برای ارائه مقدار متغیری به نام بعدی ، عنصر HTML مخفی اضافه کرده ایم. این متغیر برای اولین بار هنگام مشاهده پارامتر بعدی در درخواست ، توسط نمای ورود تنظیم می شود (به عنوان مثال ، ./next=/account/?http://127.0.0.1:8000/account/login

پارامتر بعدی باید یک URL باشد. اگر این پارامتر داده شود ، نمای ورود به سیستم Django پس از ورود موفقیت آمیز ، کاربر را به آدرس URL داده هدایت می کند.

اکنون یک الگوی logged_out.html را در داخل الگوی ثبت ایجاد کنید

دایرکتوری کنید و آنرا مانند این بنویسید:

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

{% block title %}Logged out{% endblock %}

{% block content %}
<h1>Logged out</h1>
<p>You have been successfully logged out. You can <a href="{% url
"login" %}">log-in again</a>.</p>
{% endblock %}
```

این الگوی است که پس از خروج کاربر از سایت ، جنگو نمایش می دهد.

پس از افزودن الگوهای URL و الگوهای نمایش ورود به سیستم و ورود به سیستم ، وب سایت شما برای استفاده کاربران از نمایه های احراز هویت Django برای کاربران آماده است.

اکنون وقتی کاربران وارد حساب کاربری خود می شوند، یک نمایش جدید را برای نمایش داشبورد ایجاد خواهیم کرد. پرونده برنامه حساب خود را باز کنید و کد زیر را به آن اضافه کنید:

```
from django.contrib.auth.decorators import login_required

@login_required
def dashboard(request):
    return render(request,
                  'account/dashboard.html',
                  {'section': 'dashboard'})
```

ما نمای خود را با `login_required` تزئین چارچوب تأیید اعتبار تزئین می کنیم. دکوراتور `login_required` بررسی می کند که آیا کاربر فعلی احراز هویت شده است یا خیر. اگر کاربر احراز هویت شود، نمای تزئین شده را اجرا می کند. در صورت عدم احراز هویت کاربر، کاربر را به URL ورود به سیستم با URL درخواست شده در ابتدا به عنوان یک پارامتر GET به نام بعدی تغییر مسیر می دهد. با این کار، نمای ورود پس از ورود با موفقیت، کاربران را به آدرس اینترنتی مورد نظر خود هدایت می کند. به یاد داشته باشید که برای این منظور ما یک ورودی پنهان در قالب الگوی ورود به سیستم اضافه کرده ایم.

ما همچنین یک متغیر بخش تعریف می کنیم. ما از این متغیر برای ردیابی بخش سایت استفاده می کنیم که کاربر در حال مشاهده است. نمایش های چندگانه ممکن است با همان بخش مطابقت داشته باشد. این یک روش ساده برای تعریف بخشی است که هر نمایش با آن مطابقت دارد.

اکنون، شما نیاز به ایجاد یک الگوی برای نمایش داشبورد دارید.

یک فایل جدید را درون `templates/account/` directory ایجاد کنید و آن را نامگذاری کنید

.`dashboard.html` نگاهی به این شکل دهید:

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

{% block title %}Dashboard{% endblock %}

{% block content %}
<h1>Dashboard</h1>
<p>Welcome to your dashboard.</p>
{% endblock %}
```

سپس الگوی URL زیر را برای این نمایش در پرونده urls.py برنامه حساب اضافه کنید:

```
urlpatterns = [
    # ...
    path('', views.dashboard, name='dashboard'),
]
```

پرونده settings.py پروژه خود را ویرایش کنید و کد زیر را به آن اضافه کنید:

```
LOGIN_REDIRECT_URL = 'dashboard'
LOGIN_URL = 'login'
LOGOUT_URL = 'logout'
```

تنظیمات ذکر شده در کد قبلی به شرح زیر است:

با وجود LOGIN_REDIRECT_URL در صورت عدم وجود پس از ورود به سیستم موفق به هدایت مجدد به URL برای تغییر مسیر کاربر برای ورود به سیستم (برای مثال ، نمایش ها با استفاده از login_required decorator

LOGOUT_URL: نشانی اینترنتی جهت تغییر مسیر کاربر برای خروج از سیستم

ما از نام الگوهای URL که قبلاً تعریف کرده ایم با استفاده از ویژگی نام عملکرد تابع () استفاده می کنیم. URL های هارد کد گذاری شده به جای نام های URL نیز می توانند برای این تنظیمات استفاده شوند.

اکنون ، ما پیوندهای ورود و خروج را به الگوی پایه خود اضافه خواهیم کرد تا همه چیز در کنار هم قرار بگیرد. به منظور انجام این کار ، باید مشخص کنیم که کاربر فعلی وارد سیستم شده است یا نه برای نمایش پیوند مناسب برای هر مورد. کاربر فعلی در شیء HttpRequest توسط واسطه تأیید اعتبار تنظیم شده است. می توانید با درخواست user به آن دسترسی پیدا کنید. حتی اگر کاربر تأیید نشده باشد ، یک مورد کاربر را در درخواست پیدا خواهید کرد. کاربر غیر معترض در درخواست به عنوان نمونه ای از AnonymousUser در درخواست تنظیم شده است. بهترین راه برای بررسی تأیید اعتبار کاربر فعلی ، دسترسی به ویژگی فقط خواندنی آن is_authenticated است.

الگوی base.html خود را ویرایش کنید و عنصر <div> را با شناسه هدر اصلاح کنید ، مانند این:

```
<div id="header">
  <span class="logo">Bookmarks</span>
```

```

{% if request.user.is_authenticated %}
    <ul class="menu">
        <li {% if section == "dashboard" %}class="selected"{% endif %}>
            <a href="{% url "dashboard" %}">My dashboard</a>
        </li>
        <li {% if section == "images" %}class="selected"{% endif %}>
            <a href="#">Images</a>
        </li>
        <li {% if section == "people" %}class="selected"{% endif %}>
            <a href="#">People</a>
        </li>
    </ul>
{% endif %}

<span class="user">
    {% if request.user.is_authenticated %}
        Hello {{ request.user.first_name }},
        <a href="{% url "logout" %}">Logout</a>
    {% else %}
        <a href="{% url "login" %}">Log-in</a>
    {% endif %}
</span>
</div>

```

همانطور که در کد قبلی مشاهده می کنید، فقط منوی سایت را برای کاربران معتبر نمایش می دهیم. ما همچنین بخش فعلی را بررسی می کنیم تا یک ویژگی کلاس انتخاب شده را به آیتم <a> مربوطه اضافه کنیم تا بخش فعلی را در منو با استفاده از CSS برجسته کنید. ما همچنین نام و پیوند کاربر را برای خروج از تأیید اعتبار کاربر یا پیوندی برای ورود به سیستم در غیر این صورت نمایش می دهیم.

اکنون در مرورگر خود `/account/login` را باز کنید. باید صفحه ورود را مشاهده کنید. یک نام کاربری و رمز عبور معتبر وارد کنید و بر روی دکمه Log-in کلیک کنید. باید خروجی زیر را مشاهده کنید:

Bookmarks

My dashboard

Images

People

Hello Antonio, Logout

Dashboard

Welcome to your dashboard.

مشاهده می کنید که بخش داشبورد من با CSS برجسته می شود زیرا دارای یک کلاس انتخاب شده است. از آنجا که احراز هویت کاربر است ، نام اول کاربر در سمت راست هدر نمایش داده می شود.

روی پیوند Logout کلیک کنید. باید صفحه زیر را مشاهده کنید:

Bookmarks

Log-in

Logged out

You have been successfully logged out. You can [log-in again](#).

در صفحه ذکر شده در تصویر قبلی می توانید مشاهده کنید که کاربر از سیستم خارج شده است و به همین دلیل منوی وب سایت دیگر نمایش داده نمی شود. اکنون ، پیوند در سمت راست هدر ورود به سیستم را نشان می دهد.

تغییر نمایش گذرواژه

ما همچنان به کاربران خود نیاز داریم تا پس از ورود به سایت ما بتوانند رمز عبور خود را تغییر دهند. ما نماهای تأیید هویت Django را برای تغییر رمز عبور ادغام خواهیم کرد. پرونده urls.py برنامه حساب را باز کنید و الگوهای URL زیر را به آن اضافه کنید:

```
# change password urls
path('password_change/',
      auth_views.PasswordChangeView.as_view(),
      name='password_change'),
path('password_change/done/',
      auth_views.PasswordChangeDoneView.as_view(),
      name='password_change_done'),
```

نمای PasswordChangeView برای تغییر رمز عبور از فرم استفاده می کند ، و پس از آنکه کاربر رمز عبور خود را با موفقیت تغییر داد ، نمای رمز ChangeDoneView پیام موفقیت را نشان می دهد. باید برای هر نمای الگویی ایجاد کنیم. پرونده جدیدی را در templates/registration/ directory برنامه حساب خود اضافه کنید و آنرا بنامید. کد زیر را به آن اضافه کنید:

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

{% block title %}Change you password{% endblock %}

{% block content %}
<h1>Change you password</h1>
<p>Use the form below to change your password.</p>
<form action"." method="post">
{{ form.as_p }}
<p><input type="submit" value="Change"></p>
{% csrf_token %}
</form>
{% endblock %}
```

قالب فرم شامل password_change_form.html تغییر رمز عبور است. اکنون یک فایل دیگر را در همان فهرست ایجاد کنید و آن را password_change_done.html بنامید. کد زیر را به آن اضافه کنید:

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

{% block title %}Password changed{% endblock %}

{% block content %}
<h1>Password changed</h1>
<p>Your password has been successfully changed.</p>
{% endblock %}
```

قالب password_change_done.html فقط حاوی پیام موفقیت در نمایش کاربر است که کاربر رمز عبور خود را با موفقیت تغییر داده است.

باشد، مرورگر شما را به صفحه ورود هدایت می کند. پس از تأیید اعتبار موفقیت آمیز، صفحه تغییر رمز عبور زیر را مشاهده خواهید کرد:

Change your password

Use the form below to change your password.

Old password:

New password:

- Your password can't be too similar to your other personal information.
- Your password must contain at least 8 characters.
- Your password can't be a commonly used password.
- Your password can't be entirely numeric.

New password confirmation:

CHANGE

فرم را با گزروایه فعال و رمز جدید خود پر کنید و بر روی دکمه CHANGE کلیک کنید. صفحه موفقیت زیر را مشاهده خواهید کرد:

Bookmarks

My dashboard Images People

Hello Antonio, Logout

Password changed

Your password has been successfully changed.

برای تأیید اینکه همه چیز مطابق پیش بینی شده کار کند ، از سیستم استفاده کنید و دوباره وارد سیستم شوید.

بازنشانی نمایش گذرواژه

الگوهای URL زیر را برای بازیابی رمز عبور به پرونده urls.py برنامه account اضافه کنید:

```
# reset password urls
path('password_reset/',
      auth_views.PasswordResetView.as_view(),
      name='password_reset'),
path('password_reset/done/',
      auth_views.PasswordResetDoneView.as_view(),
      name='password_reset_done'),
path('reset/<uidb64>/<token>/',
      auth_views.PasswordResetConfirmView.as_view(),
      name='password_reset_confirm'),
path('reset/done/',
      auth_views.PasswordResetCompleteView.as_view(),
      name='password_reset_complete'),
```

یک فایل جدید را در account templates/registration/ directory خود اضافه کنید و آنرا بنامید. کد زیر را به آن اضافه کنید:

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

{% block title %}Reset your password{% endblock %}

{% block content %}
<h1>Forgotten your password?</h1>
<p>Enter your e-mail address to obtain a new password.</p>
<form action"." method="post">
{{ form.as_p }}
<p><input type="submit" value="Send e-mail"></p>
{{ csrf_token }}
</form>
{% endblock %}
```

اکنون یک فایل دیگر را در همان فهرست ایجاد کنید و آنرا password_reset_email.html بنامید. کد زیر را به آن اضافه کنید:

```
Someone asked for password reset for email {{ email }}. Follow the link
below:
{{ protocol }}://{{ domain }}{{ url "password_reset_confirm" uidb64=uid
token=token %}}
Your username, in case you've forgotten: {{ user.get_username }}
```

از قالب password_reset_email.html برای ارائه ایمیل ارسال شده به کاربران برای بازنیشانی رمز عبور استفاده خواهد شد.

یک فایل دیگر را در همان فهرست ایجاد کنید و آنرا password_reset_done.html بنامید. کد زیر را به آن اضافه کنید:

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

{% block title %}Reset your password{% endblock %}

{% block content %}
    <h1>Reset your password</h1>
    <p>We've emailed you instructions for setting your password.</p>
    <p>If you don't receive an email, please make sure you've entered the
address you registered with.</p>
{% endblock %}
```

الگوی دیگری را در همان فهرست ایجاد کنید و آنرا password_reset_confirm.html بنامید. کد زیر را به آن اضافه کنید:

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

{% block title %}Reset your password{% endblock %}

{% block content %}
    <h1>Reset your password</h1>
    {% if validlink %}
        <p>Please enter your new password twice:</p>
        <form action="." method="post">
            {{ form.as_p }}
            {% csrf_token %}
            <p><input type="submit" value="Change my password" /></p>
        </form>
    {% else %}

```

```
<p>The password reset link was invalid, possibly because it has  
already been used. Please request a new password reset.</p>  
{% endif %}  
{% endblock %}
```

ما بررسی می کنیم که آیا لینک ارائه شده معتبر است یا خیر. نمای PasswordResetConfirmView این متغیر را تنظیم کرده و آن را در قالب الگوی password_reset_confirm.html قرار می دهد. اگر پیوند معتبر باشد ، فرم تنظیم مجدد رمز عبور کاربر را نمایش می دهیم.

یک الگوی دیگر ایجاد کنید و آن را password_reset_complete.html بنامید. کد زیر را در آن وارد کنید:

```
{% extends "base.html" %}  
  
{% block title %}Password reset{% endblock %}  
  
{% block content %}  
    <h1>Password set</h1>  
    <p>Your password has been set. You can <a href="{% url "login" %}">log in  
now</a></p>  
{% endblock %}
```

در آخر ، الگوی registration/login.html برنامه حساب را ویرایش کنید و بعد از عنصر `<form>` کد زیر را اضافه کنید:

```
<p><a href="{% url "password_reset" %}">Forgotten your  
password?</a></p>
```

اکنون در مرورگر خود <http://127.0.0.1:8000/account/login> را باز کنید و بر روی رمز فراموش شده خود کلیک کنید؟ ارتباط دادن. باید صفحه زیر را مشاهده کنید:



در این مرحله ، باید پیکربندی SMTP را به پرونده settings.py پروژه خود اضافه کنید تا Django بتواند ایمیل بفرستد. شما یاد گرفتید که چگونه می توانید تنظیمات ایمیل را به بخش خود اضافه کنید و در فصل 2 ، وبلاگ خود را با ویژگی های پیشرفته بهبود بخشید. با این حال ، در حین توسعه ، می توانید به جای ارسال آنها از طریق سرور SMTP ، Django را برای نوشتن ایمیل به خروجی استاندارد پیکربندی کنید.

یک پس زمینه ایمیل برای نوشتن ایمیل به کنسول فراهم می کند.

پرونده settings.py پروژه خود را ویرایش کرده و خط زیر را اضافه کنید:

```
EMAIL_BACKEND = 'django.core.mail.backends.console.EmailBackend'
```

تنظیم EMAIL_BACKEND نشانگر کلاس مورد استفاده برای ارسال ایمیل است. به مرورگر خود بازگردید ، آدرس ایمیل کاربر موجود را وارد کنید و بر روی دکمه SEND E-MAIL کلیک کنید. باید صفحه زیر را مشاهده کنید:



Reset your password

We've emailed you instructions for setting your password.

If you don't receive an email, please make sure you've entered the address you registered with.

نگاهی به کنسولی که در آن سرور توسعه وجود دارد ، نگاهی بیندازید. ایمیل تولید شده را به شرح زیر مشاهده خواهید کرد:

```
Content-Type: text/plain; charset="utf-8"
MIME-Version: 1.0
Content-Transfer-Encoding: 7bit
Subject: Password reset on 127.0.0.1:8000
From: webmaster@localhost
To: user@domain.com
Date: Fri, 15 Dec 2017 14:35:08 -0000
Message-ID: <20150924143508.62996.55653@zenx.local>
```

Someone asked for password reset for email user@domain.com. Follow the link below:
<http://127.0.0.1:8000/account/reset/MQ/45f-9c3f30caaf523055fcc/>
Your username, in case you've forgotten: zenx

ایمیل با استفاده از الگوی password_reset_email.html که قبلاً ایجاد کردیم ارائه می شود. URL برای تنظیم مجدد گذرواژتان حاوی نشانه‌ای است که توسط Django بصورت پویا تولید شده است. URL را کپی کرده و در مرورگر خود باز کنید. باید صفحه زیر را مشاهده کنید:

The screenshot shows a web browser window with a green header bar. On the left is a 'Bookmarks' button, and on the right is a 'Log-in' button. Below the header, the main content area has a heading 'Reset your password'. A sub-instruction 'Please enter your new password twice:' is followed by two input fields. The first input field is labeled 'New password:' and the second is labeled 'New password confirmation:'. Below these fields is a green button labeled 'CHANGE MY PASSWORD'. Above the input fields, there is a bulleted list of password requirements:

- Your password can't be too similar to your other personal information.
- Your password must contain at least 8 characters.
- Your password can't be a commonly used password.
- Your password can't be entirely numeric.

صفحه تنظیم گذرواژه جدید با الگوی password_reset_confirm.html مطابقت دارد. رمزعبور جدیدی را وارد کرده و بر روی دکمه CHANGE MY PASSWORD کلیک کنید. Django یک رمزعبور رمزگذاری شده جدید ایجاد می کند و آن را در بانک اطلاعات ذخیره می کند. صفحه موفقیت زیر را مشاهده خواهید کرد:

Password set

Your password has been set. You can [log in now](#)

اکنون می توانید با استفاده از رمز جدید خود وارد حساب کاربری خود شوید. از هر نشانه ای برای تنظیم رمز عبور جدید فقط یک بار می توان استفاده کرد. اگر پیوندی را که دوباره دریافت کردید باز کنید ، پیامی دریافت خواهید کرد که نشان آن نامعتبر است. شما دیدگاه های چارچوب تأیید هویت Django را در پروژه خود ادغام کرده اید. این نمایه برای بیشتر موارد مناسب است. با این وجود ، اگر به رفتاری متفاوت نیاز دارید ، می توانید نظرات خود را ایجاد کنید.

همچنانی الگوهای URL تأیید هویت را که ما تازه ایجاد کردیم فراهم می کند. می توانید الگوهای URL تأیید اعتبار را که ما به پرونده `urls.py` برنامه حساب اضافه کردیم ، اظهار نظر کنید و در عوض `django.contrib.auth.urls` را شامل کنید:

```
from django.urls import path, include
# ...

urlpatterns = [
    # ...
    path('', include('django.contrib.auth.urls')),
]
```

می توانید الگوهای URL تأیید اعتبار موجود در آن را مشاهده کنید

<https://github.com/django/django/blob/stable/2.0.x/django/contrib/auth/urls.py>

ثبت نام کاربر و پروفایل کاربر

کاربران موجود اکنون می توانند وارد سیستم شوند ، از سیستم خارج شوند ، رمز عبور خود را تغییر دهند و رمز ورود خود را دوباره تنظیم کنند. اکنون ، ما نیاز به ایجاد یک نمایش داریم تا بازدید کنندگان بتوانند یک حساب کاربری ایجاد کنند.

ثبت نام کاربر

باید یک نمایش ساده ایجاد کنیم تا ثبت نام کاربر در وب سایت ما امکان پذیر باشد.

در ابتدا ، ما باید یک فرم ایجاد کنیم تا کاربر بتواند نام کاربری را وارد کند ،

نام واقعی آنها و رمز عبور پرونده forms.py را که در فهرست برنامه کاربردی حساب قرار دارد ویرایش کنید و کد زیر را به آن اضافه کنید:

```
from django.contrib.auth.models import User

class UserRegistrationForm(forms.ModelForm):
    password = forms.CharField(label='Password',
                                widget=forms.PasswordInput)
    password2 = forms.CharField(label='Repeat password',
                                widget=forms.PasswordInput)

    class Meta:
        model = User
        fields = ('username', 'first_name', 'email')

    def clean_password2(self):
        cd = self.cleaned_data
        if cd['password'] != cd['password2']:
            raise forms.ValidationError('Passwords don\'t match.')
        return cd['password2']
```

ما یک مدل برای مدل کاربر ایجاد کرده ایم. در فرم ما ، فقط نام کاربری ، first_name و زمینه های ایمیل مدل را درج می کنیم. این زمینه ها بر اساس زمینه های مدل مربوطه ، تأیید می شوند.

به عنوان مثال ، اگر کاربر نام کاربری را که قبلًا وجود داشته باشد انتخاب کند ، خطای اعتبار سنجی را دریافت می کند زیرا نام کاربری فیلدی است که با منحصر به فرد = True تعریف شده است. ما دو فیلد اضافی اضافه کرده ایم - گذرواژه و کلمه عبور - برای کاربران برای تنظیم رمز عبور و تأیید آن. ما یک روش Clean_password2 تعریف کرده ایم تا رمز دوم را در برابر اولین مورد بررسی کنیم و اجازه ندهیم در صورت عدم تطابق کلمات ، فرم را تأیید کند.

این بررسی هنگامی انجام می شود که فرم مورد نظر را با استفاده از روش is_valid اعتبار دهیم. شما می توانید یک روش <clean_<fieldname>> را برای هر یک از روش های خود ارائه دهید

برای پاک کردن مقدار یا افزایش خطاهای اعتبار سنجی فرم برای یک قسمت خاص ، زمینه را ایجاد کنید. فرم ها همچنین شامل یک روش کلی تمیز () برای اعتبار سنجی کل فرم است که برای اعتبار سنجی زمینه هایی که به یکدیگر وابسته هستند ، مفید است. Django همچنین فرم کاربر UserCreationForm را فراهم می کند که می توانید از آن استفاده کنید ، که در django.contrib.auth.forms سکونت دارد و بسیار شبیه به روشنی است که ما ایجاد کرده ایم.

پرونده views.py برنامه حساب را ویرایش کنید و کد زیر را به آن اضافه کنید:

```

from .forms import LoginForm, UserRegistrationForm

def register(request):
    if request.method == 'POST':
        user_form = UserRegistrationForm(request.POST)
        if user_form.is_valid():
            # Create a new user object but avoid saving it yet
            new_user = user_form.save(commit=False)
            # Set the chosen password
            new_user.set_password(
                user_form.cleaned_data['password'])
            # Save the User object
            new_user.save()
            return render(request,
                          'account/register_done.html',
                          {'new_user': new_user})
    else:
        user_form = UserRegistrationForm()
    return render(request,
                  'account/register.html',
                  {'user_form': user_form})

```

نمای ایجاد حساب کاربری بسیار ساده است. به جای صرفه جویی در رمز عبور خام وارد شده توسط کاربر، از روش set_password از مدل کاربر استفاده می کنیم که برای رمزگذاری به دلایل ایمنی از رمزگذاری استفاده می کند.
اکنون پرونده urls.py برنامه حساب خود را ویرایش کرده والگوی URL زیر را اضافه کنید:

```
path('register/', views.register, name='register'),
```

سرانجام ، الگوی جدیدی را در نام دایرکتوری account/template ایجاد کنید که آن را در register.html ثبت کنید.
ایجاد کنید و به صورت زیر ظاهر کنید:

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

{% block title %}Create an account{% endblock %}

{% block content %}
    <h1>Create an account</h1>
    <p>Please, sign up using the following form:</p>
    <form action"." method="post">
        {{ user_form.as_p }}
        {% csrf_token %}
        <p><input type="submit" value="Create my account"></p>
    </form>
{% endblock %}
```

یک فایل قالب را در همان فهرست اضافه کنید و itregister_done.html را نامگذاری کنید. کد زیر را به آن اضافه کنید:

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

{% block title %}Welcome{% endblock %}

{% block content %}
    <h1>Welcome {{ new_user.first_name }}!</h1>
    <p>Your account has been successfully created. Now you can <a href="{%
    "login" %}">log in</a>.</p>
{% endblock %}
```

اکنون در مرورگر خود <http://127.0.0.1:8000/account/register> را باز کنید. صفحه ثبت نام خود را مشاهده خواهید کرد:

Create an account

Please, sign up using the following form:

Username:

Required. 150 characters or fewer. Letters, digits and @/./+/_- only.

First name:

Email address:

Password:

Repeat password:

[CREATE MY ACCOUNT](#)

جزئیات مربوط به یک کاربر جدید را پر کنید و بر روی دکمه CREATE MY ACCOUNT کلیک کنید. اگر همه زمینه ها معتبر باشند، کاربر ایجاد می شود و پیام موفقیت زیر را دریافت می کنید:

Welcome Paloma!

Your account has been successfully created. Now you can [log in](#).

روی پیوند ورود به سیستم کلیک کنید و نام کاربری و رمز عبور خود را وارد کنید تا تأیید کنید که می توانید به حساب خود دسترسی پیدا کنید.

اکنون می توانید پیوندی برای ثبت نام در الگوی ورود خود اضافه کنید. الگوی ثبت نام [registration/login.html](#) ویرایش کنید. نگاهی به خط زیر بیندازید:

```
<p>Please, use the following form to log-in:</p>
```

آن را با موارد زیر جایگزین کنید:

```
<p>Please, use the following form to log-in. If you don't have an account <a href="#">register here</a></p>
```

ما صفحه ثبت نام را از صفحه ورود به سیستم در دسترس قرار دادیم.

گسترش مدل کاربر

هنگامی که شما باید با حساب های کاربری سروکار داشته باشید ، متوجه می شوید که مدل کاربر چارچوب احراز هویت Django برای موارد رایج مناسب است. با این حال ، مدل کاربر دارای زمینه های بسیار اساسی است. ممکن است مایل باشید مدل کاربر را گسترش داده و داده های اضافی را درج کنید. بهترین راه برای انجام این کار با ایجاد یک مدل پروفایل است که شامل تمام زمینه های اضافی و رابطه یک به یک با مدل کاربر Django است.

پرونده model.py برنامه حساب خود را ویرایش کنید و کد زیر را به آن اضافه کنید:

```
from django.db import models
from django.conf import settings

class Profile(models.Model):
    user = models.OneToOneField(settings.AUTH_USER_MODEL,
                               on_delete=models.CASCADE)
    date_of_birth = models.DateField(blank=True, null=True)
    photo = models.ImageField(upload_to='users/%Y/%m/%d/',
                             blank=True)

    def __str__(self):
        return 'Profile for user {}'.format(self.user.username)
```

کاربر به شما امکان می دهد پروفایل های خود را با کاربران مرتبط کنید. ما برای پارامتر on_delete از زمینه one-to-one استفاده می کنیم تا هنگام حذف کاربر ، پروفایل مربوط به آن نیز حذف شود. قسمت عکس یک فیلد CASCADE است. برای رسیدگی به تصاویر باید کتابخانه Pillow را نصب کنید. Pillow را با اجرای دستور زیر در پوسته خود نصب کنید:

```
pip install Pillow==5.1.0
```

برای سرویس دهی جنگو به پرونده های رسانه ای آپلود شده توسط کاربران با سرور توسعه ، تنظیمات زیر را به پرونده تنظیمات.py پروژه خود اضافه کنید:

```
MEDIA_URL = '/media/'  
MEDIA_ROOT = os.path.join(BASE_DIR, 'media/')
```

MEDIA_URL اصلی برای ارائه پرونده های رسانه ای است که توسط کاربران بارگذاری می شود و MEDIA_ROOT مسیر محلی است که در آن ساکن هستند. ما مسیر را بطور پویا نسبت به مسیر پروژه خود می سازیم تا کدهای خود را عمومی تر کنند.

اکنون فایل اصلی urls.py پروژه بوکمارک ها را ویرایش کرده و کد را به شرح زیر تغییر دهید:

```
from django.contrib import admin
from django.urls import path, include
from django.conf import settings
from django.conf.urls.static import static

urlpatterns = [
    path('admin/', admin.site.urls),
    path('account/', include('account.urls')),
]

if settings.DEBUG:
    urlpatterns += static(settings.MEDIA_URL,
                          document_root=settings.MEDIA_ROOT)
```

به این ترتیب ، سرور توسعه Django وظیفه ارائه پرونده های رسانه ای را در حین توسعه (یعنی وقتی تنظیم DEBUG روی True تنظیم شده است) خواهد بود.

پوسته را باز کرده و دستور زیر را برای ایجاد انتقال پایگاه داده برای مدل جدید اجرا کنید:

```
python manage.py makemigrations
```

شما خروجی زیر را دریافت خواهید کرد:

```
Migrations for 'account':  
  account/migrations/0001_initial.py  
    - Create model Profile
```

بعد ، پایگاه داده را با دستور زیر همگام سازی کنید:

```
python manage.py migrate
```

خروجی را مشاهده خواهید کرد که شامل خط زیر است:

```
Applying account.0001_initial... OK
```

پرونده admin.py برنامه حساب را ویرایش کنید و مدل پروفایل را در سایت مدیریت ثبت کنید ، مانند این:

```
from django.contrib import admin  
from .models import Profile  
  
@admin.register(Profile)  
class ProfileAdmin(admin.ModelAdmin):  
    list_display = ['user', 'date_of_birth', 'photo']
```

سرور توسعه را با استفاده از عامل اصلی Python management.py اجرا کنید <http://127.0.0.1:8000/admin> T. در مرورگر خود باز کرده و باز کنید. حال ، شما باید بتوانید مدل پروفایل ها را در سایت مدیریت پروژه خود مشاهده کنید ، به

شرح زیر است:

ACCOUNT

Profiles

+ Add

Change

اکنون ، ما به کاربران اجازه خواهیم داد نمایه خود را در وب سایت ویرایش کنند. فرم های واردات و مدل زیر را به پرونده فرم.py برنامه حساب اضافه کنید:

```
from .models import Profile

class UserEditForm(forms.ModelForm):
    class Meta:
        model = User
        fields = ('first_name', 'last_name', 'email')

class ProfileEditForm(forms.ModelForm):
    class Meta:
        model = Profile
        fields = ('date_of_birth', 'photo')
```

این فرم ها به شرح زیر است:

UserEditForm: با این کار به کاربران امکان ویرایش نام ، نام خانوادگی و ایمیل خود را می دهد ، که اینها ویژگی های مدل کاربر سازنده جنگو هستند.

ProfileEditForm: این به شما امکان می دهد تا داده های نمایه ای را که در مدل Profile سفارشی ذخیره می کنیم ، ویرایش کنند. کاربران می توانند تاریخ تولد خود را ویرایش کرده و تصویری را برای پروفایل خود بارگذاری کنند.

پرونده views.py برنامه حساب را ویرایش کنید و مدل Profile را مانند این وارد کنید:

```
from .models import Profile
```

سپس خطوط زیر را به نمای ثبت در زیر new_user.save () اضافه کنید:

```
# Create the user profile  
Profile.objects.create(user=new_user)
```

وقتی کاربران در سایت ما ثبت نام می کنند ، نمایه ای خالی در ارتباط با آنها ایجاد خواهیم کرد. شما باید با استفاده از سایت مدیریت برای کاربرانی که قبلاً ایجاد کرده اید ، یک شیء Profile بصورت دستی ایجاد کنید.

اکنون به کاربران اجازه خواهیم داد نمایه خود را ویرایش کنند کد زیر را به همان پرونده اضافه کنید:

```
from .forms import LoginForm, UserRegistrationForm, \
                  UserEditForm, ProfileEditForm

@login_required
def edit(request):
    if request.method == 'POST':
        user_form = UserEditForm(instance=request.user,
                               data=request.POST)
        profile_form = ProfileEditForm(
            instance=request.user.profile,
            data=request.POST,
            files=request.FILES)
        if user_form.is_valid() and profile_form.is_valid():
            user_form.save()
            profile_form.save()
    else:
        user_form = UserEditForm(instance=request.user)
        profile_form = ProfileEditForm(
            instance=request.user.profile)
    return render(request,
                  'account/edit.html',
                  {'user_form': user_form,
                   'profile_form': profile_form})
```

ما از دکوراتور login_required استفاده می کنیم زیرا کاربران باید ویرایش نمایه خود را تأیید کنند. در این حالت ، ما از دو شکل مدل استفاده می کنیم: UserEditForm برای ذخیره داده های مدل کاربر داخلی و ProfileEditForm برای ذخیره داده های پروفایل اضافی در مدل Profile سفارشی. برای تأیید اعتبار داده های ارسالی ، روش is_valid () هر دو فرم را اجرا می کنیم. اگر هر دو فرم حاوی داده های معتبر باشند ، ما هر دو فرم را ذخیره می کنیم و با استفاده از روش save () برای به روزرسانی اشیاء مربوطه در بانک اطلاعاتی استفاده می کنیم.

الگوی URL زیر را به پرونده urls.py برنامه حساب اضافه کنید:

```
path('edit/', views.edit, name='edit'),
```

در آخر ، الگویی برای این نمای در `templates/account/edit.html` ایجاد کنید و آنرا بنامید. کد زیر را به آن اضافه کنید:

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

{% block title %}Edit your account{% endblock %}

{% block content %}
<h1>Edit your account</h1>
<p>You can edit your account using the following form:</p>
<form action="." method="post" enctype="multipart/form-data">
    {{ user_form.as_p }}
    {{ profile_form.as_p }}
    {% csrf_token %}
    <p><input type="submit" value="Save changes"></p>
</form>
{% endblock %}
```

ما برای فعال کردن بارگذاری پرونده ، شامل "enctype = "multipart / form-data" هستیم. ما از فرم HTML برای ارسال فرم های `profile_form` و `user_form` استفاده می کنیم.

کاربر جدیدی را ثبت کنید و `http://127.0.0.1:8000/account/edit/` را باز کنید. باید صفحه زیر را مشاهده کنید:

Edit your account

You can edit your account using the following form:

First name:

Paloma

Last name:

Melé

Email address:

paloma@zenxit.com

Date of birth:

1981-04-14

Photo:

Choose File no file selected

SAVE CHANGES

اکنون می توانید صفحه داشبورد را نیز ویرایش کرده و پیوندهایی را به پروفایل ویرایش وارد کرده و صفحات رمز عبور را تغییر دهید. الگوی account/dashboard.html را باز کنید:

```
<p>Welcome to your dashboard.</p>
```

خط قبل را با خط زیر جایگزین کنید:

```
<p>Welcome to your dashboard. You can <a href="{% url "edit" %}">edit your profile</a> or <a href="{% url "password_change" %}">change your password</a>.</p>
```

کاربران اکنون می توانند برای ویرایش نمایه خود از داشبورد خود به این فرم دسترسی پیدا کنند.
را در مرورگر خود باز کنید و پیوند جدید را برای ویرایش نمایه کاربر آزمایش کنید.

Dashboard

Welcome to your dashboard. You can [edit your profile](#) or [change your password](#).

استفاده از یک مدل کاربر سفارشی

همچنین راهی برای جایگزینی کل مدل کاربر با مدل دلخواه خود ارائه می دهد. کلاس کاربر شما باید از کلاس AbstractUser Django استفاده کند ، که اجرایی کامل کاربر پیش فرض را به عنوان یک مدل انتزاعی فراهم می کند. می توانید اطلاعات بیشتر در مورد این روش را در <https://docs.djangoproject.com/en/2.0/topics/auth/customizing/#substituting-a-custom-user-model> بخوانید.

استفاده از یک مدل کاربر سفارشی ، انعطاف پذیری بیشتری به شما می دهد ، اما ممکن است منجر به یکپارچه سازی سخت تر با برنامه های قابل افزایشی شود که با مدل کاربر auth Django تعامل دارند.

استفاده از چارچوب پیام ها

هنگامی که به کاربران اجازه می دهدید با سیستم عامل شما ارتباط برقرار کنند ، موارد بسیاری وجود دارد که ممکن است بخواهید آنها را از نتیجه اقدامات خود به آنها اطلاع دهید. Django فریم ورک داخلی دارد که به شما امکان می دهد اعلان های یک بار را به کاربران خود نمایش دهید.

چارچوب پیام ها در django.contrib.messages قرار دارد و در هنگام ایجاد پروژه های جدید با استفاده از پروژه شروع Python ، در لیست پیش فرض INSTALLED_APPS پرونده تنظیمات.py قرار می گیرد. توجه داشته باشید که پرونده تنظیمات شما حاوی واسطه ای به نام django.contrib.messages.middleware.MessageMiddleware در تنظیمات MIDDLEWARE است.

چارچوب پیامها راهی ساده برای افزودن پیام به کاربران ارائه می دهد. پیام ها به طور پیش فرض در یک کوکی ذخیره می شوند (بازگشت به جلسه جلسه) ، و در درخواست بعدی که کاربر انجام می دهد نمایش داده می شوند. می توانید با وارد کردن ماژول پیام ها و اضافه کردن پیام های جدید با کلید های میانبر ساده ، از چارچوب پیام ها در نمای خود استفاده کنید:

```
from django.contrib import messages
messages.error(request, 'Something went wrong')
```

می توانید با استفاده از روش add_message () یا هر یک از روش های میانبر زیر پیام های جدید ایجاد کنید:

موفقیت (): پیام های موفقیت آمیز بعد از موفقیت در یک عمل نمایش داده می شوند

اطلاعات (): پیامهای اطلاع رسانی

هشدار (): هنوز چیزی نتوانسته است شکست بخورد ، اما ممکن است بی نهایت شکست بخورد

خطا (): یک عمل موفقیت آمیز نبود ، یا کاری انجام نشد

اشکال زدایی (): پیام های اشکال زدایی که در یک محیط تولید حذف یا نادیده گرفته می شوند

بیایید به پلتفرم خود پیام اضافه کنیم. از آنجا که چارچوب پیام ها در سطح جهانی برای پروژه اعمال می شود ، می توانیم پیام های کاربر را در الگوی پایه خود نمایش دهیم. الگوی base.html برنامه حساب را باز کنید و کد زیر را بین عنصر <div> با شناسه هدر و عنصر <div> با شناسه محتوا اضافه کنید:

```
{% if messages %}
<ul class="messages">
    {% for message in messages %}
        <li class="{{ message.tags }}>
            {{ message|safe }}
            <a href="#" class="close">x</a>
        </li>
    {% endfor %}
</ul>
{% endif %}
```

چارچوب پیامها شامل متن پردازنده django.contrib.messages.context_processors.messages است که یک متغیر پیام را به متن درخواست اضافه می کند. می توانید آن را در لیست زمینه _ پردازنده های تنظیم TEMPLATES پروژه خود بیابید. شما می توانید از این متغیر در قالب های خود استفاده کنید تا تمام پیام های موجود به کاربر نمایش داده شود.

حال ، اجازه دهید نمای ویرایش خود را اصلاح کنیم تا از چارچوب پیام استفاده کنیم. پرونده views.py برنامه حساب را ویرایش کنید ، پیام های وارد شده را وارد کنید و نمای ویرایش را به شرح زیر بنویسید:

```
from django.contrib import messages

@login_required
```

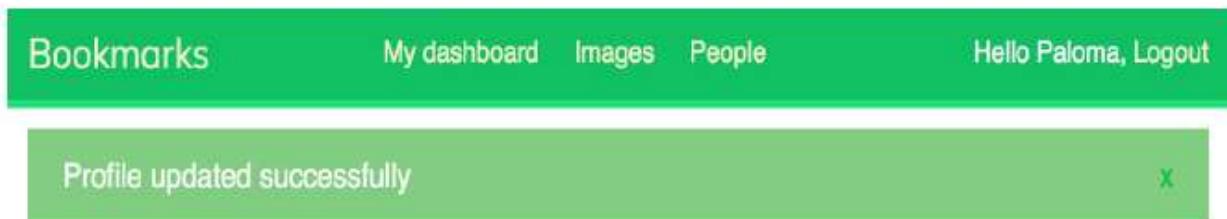
```

def edit(request):
    if request.method == 'POST':
        # ...
        if user_form.is_valid() and profile_form.is_valid():
            user_form.save()
            profile_form.save()
            messages.success(request, 'Profile updated \'\\
                \'successfully')
        else:
            messages.error(request, 'Error updating your profile')
    else:
        user_form = UserEditForm(instance=request.user)
        # ...

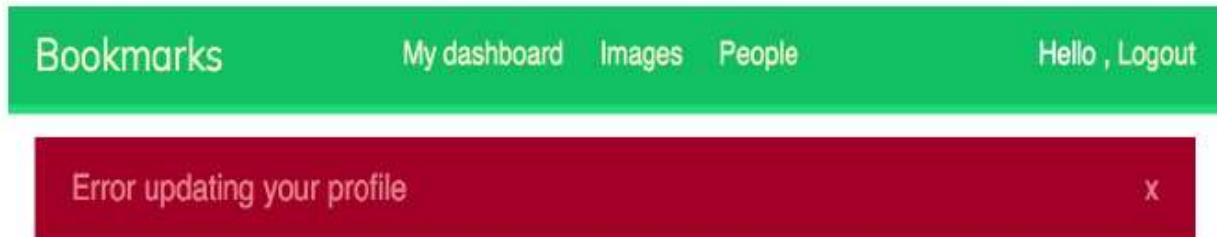
```

وقتی کاربر با موفقیت پروفایل خود را به روز کرد ، یک پیام موفقیتی اضافه می کنیم. اگر هر یک از اشکال حاوی داده های نامعتبر است ، به جای آن یک پیام خطأ اضافه می کنیم.

را در مرورگر خود باز کرده و نمایه خود را ویرایش کنید. وقتی نمایه با موفقیت به روز شد ، باید پیام زیر را مشاهده کنید:



هنگامی که داده ها معتبر نیست ، به عنوان مثال ، با استفاده از یک تاریخ نادرست قالب بندی شده برای تاریخ تولد ، باید پیام زیر را مشاهده کنید:



ایجاد یک پس زمینه تأیید هویت سفارشی

جنگو به شما اجازه می دهد تا در مقابل منابع مختلف تأیید اعتبار کنید. تنظیم AUTHENTICATION_BACKENDS شامل لیست اعتبارهای تأیید اعتبار برای پروژه شما است. به طور پیش فرض ، این تنظیم به شرح زیر است:

```
[ 'django.contrib.auth.backends.ModelBackend' ]
```

پیش فرض با استفاده از مدل کاربر django.contrib.auth کاربران را در برابر دیتابیس احراز هویت می کند. این مناسب با بیشتر پروژه های شما خواهد بود. با این حال ، می توانید برای تأیید اعتبار کاربران خود در برابر منابع دیگر ، مانند یک فهرست دایرکتوری LDAP یا هر سیستم دیگر ، دارای بک های سفارشی ایجاد کنید.

می توانید اطلاعات بیشتر در مورد شخصی سازی تأیید اعتبار را در <https://docs.djangoproject.com/fa/2.0/topics/auth/customizing/#otherauthentication-s> منابع بخوانید.

هربار که از عملکرد تأیید صحت () استفاده می کنید ، Django سعی می کند کاربر را در برابر هر کدام از پشتی های تعریف شده در AUTHENTICATION_BACKENDS یک به یک تأیید کند ، تا یک از آنها با موفقیت کاربر را تأیید کند. فقط در صورت عدم تأیید اعتبار همه ی پشتیبان ها ، کاربر در سایت شما تأیید نمی شود.

جنگو روشهای ساده برای تعریف شناسنامه شخصی خود ارائه می دهد. یک پس زمینه تأیید اعتبار کلاس است که دو روش زیر را ارائه می دهد:

تأیید اعتبار () : به عنوان پارامتر ، شیء درخواست و اعتبار کاربر را می گیرد. اگر اعتبار آن معتبر باشد ، و یا در غیر این صورت مجبور به بازگشت یک شی کاربر است که مطابق آن اعتبارنامه باشد. پارامتر درخواست یک شیء HttpRequest است ، یا در صورت عدم ارائه اعتبار () ، هیچ یک از آنها نیست.

get_user () : یک پارامتر شناسه کاربر را در اختیار می گیرد و باید یک کاربر را برگرداند.

ایجاد یک پس زمینه احراز هویت سفارشی همانند نوشتن یک کلاس پایتون است که هر دو روش را پیاده سازی می کند. ما یک پس زمینه تأیید اعتبار ایجاد خواهیم کرد تا به کاربران اجازه دهیم در سایت ما با استفاده از آدرس ایمیل خود به جای نام کاربری خود تأیید اعتبار کنند.

یک پرونده جدید را درون فهرست برنامه کاربردی حساب خود ایجاد کنید و آن را `navicationication.py` بنویسید. کد زیر را به آن اضافه کنید:

```
from django.contrib.auth.models import User

class EmailAuthBackend(object):
    """
    Authenticate using an e-mail address.
    """
    def authenticate(self, request, username=None, password=None):
        try:
            user = User.objects.get(email=username)
            if user.check_password(password):
                return user
            return None
        except User.DoesNotExist:
            return None

    def get_user(self, user_id):
        try:
            return User.objects.get(pk=user_id)
        except User.DoesNotExist:
            return None
```

کد قبلی یک پس زمینه تأیید هویت ساده است. روش تأیید اعتبار یک شیء درخواست و نام کاربری و پارامترهای اختیاری رمز عبور را دریافت می کند. ما می توانیم از پارامترهای مختلف استفاده کنیم ، اما از یک نام کاربری و رمز عبور استفاده می کنیم تا باطن ما با نمایهای چارچوب تأیید صحت کار کند. کد قبلی به شرح زیر عمل می کند:

ما سعی می کنیم با استفاده از آدرس ایمیل داده شده کاربر را بازیابی کنیم و رمز عبور را با استفاده از روش داخلی ساخته شده `check_password()` مدل کاربر بررسی کنیم. این روش برای مقایسه رمز عبور داده شده با رمز عبور ذخیره شده در دیتابیس ، از hashing password استفاده می کند.

از طریق شناسه تعیین شده در پارامتر user_id کاربر را دریافت می‌کنیم. Django از پس زمینه ای که کاربر را تأیید کرده است برای بازیابی شی کاربر برای مدت زمان جلسه کاربر استفاده می‌کند.

پرونده settings.py پروژه خود را ویرایش کرده و تنظیمات زیر را اضافه کنید:

```
AUTHENTICATION_BACKENDS = [  
    'django.contrib.auth.backends.ModelBackend',  
    'account.authentication.EmailAuthBackend',  
]
```

در تنظیم قبلی ، پیش فرض ModelBackend را که برای احراز هویت با نام کاربری و رمزعبور استفاده می شود ، نگه داشته ایم و دارای پس زمینه احراز هویت مبتنی بر ایمیل خود هستیم. اکنون در مرورگر خود <http://127.0.0.1:8000/account/login> را باز کنید. به یاد داشته باشید که جنگو سعی خواهد کرد تا کاربر را در برابر هر یک از باطن ها تأیید کند ، بنابراین اکنون باید بتوانیم با استفاده از نام کاربری یا حساب ایمیل خود وارد یکپارچه شویم. اعتبارنامه کاربر با استفاده از سابقه تأیید اعتبار ModelBackend بررسی می شود و در صورت عدم بازگشت کاربر ، اعتبار نامه ها با استفاده از سنت سفارشی EmailAuthBackend ما بررسی می شود.

اضافه کردن احراز هویت اجتماعی به سایت شما

همچنین ممکن است بخواهید با استفاده از خدماتی مانند Facebook ، Twitter یا Google ، احراز هویت اجتماعی را به سایت خود اضافه کنید. Python Social Auth ماثول Python است که فرایند افزودن احراز هویت اجتماعی به وب سایت ما را ساده می کند. با استفاده از این ماثول ، می توانید با استفاده از حساب سایر خدمات ، به کاربران خود وارد وب سایت شوید.

می توانید کد این ماثول را در <https://github.com/python-social-auth> پیدا کنید.

این ماثول دارای پشتیبان های تأیید اعتبار برای چهار چوب های مختلف پایتون ، از جمله جنگو است. برای نصب بسته Django از طریق پیپ ، کنسول را باز کرده و دستور زیر را اجرا کنید:

```
pip install social-auth-app-django==2.1.0
```

سپس ، در پرونده تنظیمات.py پروژه خود ، social_django را به تنظیمات INSTALLED_APPS اضافه کنید:

```
INSTALLED_APPS = [  
    #...  
    'social_django',  
]
```

این برنامه پیش فرض برای اضافه کردن python-social-auth به پروژه های Django است. اکنون ، دستور زیر را برای همگام سازی مدل های python-social-auth با پایگاه داده خود اجرا کنید:

```
python manage.py migrate
```

باید ببینید که مهاجرت برای برنامه پیش فرض به شرح زیر اعمال می شود:

```
Applying social_django.0001_initial... OK  
Applying social_django.0002_add_related_name... OK  
...  
Applying social_django.0008_partial_timestamp... OK
```

شامل خدمات پس زمینه ای برای چندین سرویس است. لیستی از همه پشتیبان‌ها را می‌توانید در اینجا مشاهده کنید

.<https://python-social-auth.readthedocs.io/fa/latest/backends/index.html#supported-backends>

ما شامل نسخه پشتیبان تأیید اعتبار برای Facebook و Twitter و Google خواهیم بود.

شما باید الگوهای URL ورود به سیستم اجتماعی را به پروژه خود اضافه کنید. پرونده اصلی urls.py پروژه بوک مارک‌ها را باز کنید و الگوهای URL social_django را به شرح زیر وارد کنید:

```
urlpatterns = [
    path('admin/', admin.site.urls),
    path('account/', include('account.urls')),
    path('social-auth/',
        include('social_django.urls', namespace='social')),
]
```

چندین سرویس اجتماعی پس از احراز هویت موفق ، امكان هدایت مجدد کاربران به 127.0.0.1 یا localhost را ندارند. به منظور ایجاد تأیید اعتبار اجتماعی ، به یک دامنه احتیاج دارید. برای رفع این مشکل ، تحت لینوکس یا macOS ، پرونده / etc / hosts خود را ویرایش کرده و خط زیر را به آن اضافه کنید:

```
127.0.0.1 mysite.com
```

این به رایانه شما می‌گوید که نام میزبان mysite.com را به دستگاه خودتان نشان دهد. اگر از Windows استفاده می‌کنید ، پرونده میزبان‌های شما در واقع است

.C:\Windows\System32\Drivers\etc\host

برای تأیید اینکه تغییر مسیر میزبان شما کار کرده است ، سرور توسعه را با اجرای management.py server را در مرورگر خود باز کنید. خطای زیر را مشاهده خواهید کرد:

DisallowedHost at /account/login/

Invalid HTTP_HOST header: 'mysite.com:8000'. You may need to add 'mysite.com' to ALLOWED_HOSTS.

جنگو با استفاده از تنظیمات ALLOWED_HOSTS میزانی که قادر به ارائه برنامه شما هستند را کنترل می کند. این یک اقدام امنیتی برای جلوگیری از حملات هدر میزان HTTP است. جنگو فقط به میزان های موجود در این لیست اجازه می دهد تا برنامه را ارائه دهند. می توانید درباره تنظیمات ALLOWED_HOSTS در <https://docs.djangoproject.com/en/2.0/ref/settings/#allowed-hosts> بیشتر بدانید.

پرونده تنظیمات.py پروژه خود را ویرایش کنید و تنظیمات ALLOWED_HOSTS را به شرح زیر ویرایش کنید:

```
ALLOWED_HOSTS = ['mysite.com', 'localhost', '127.0.0.1']
```

علاوه بر میزان mysite.com ، ما صریحا localhost و 127.0.0.1 را نیز شامل می شود.

ما این کار را می کنیم تا بتوانیم از طریق localhost به سایت دسترسی پیدا کنیم ، که رفتار پیش فرض جنگو است وقتی DEBUG صحیح است و ALLOWED_HOSTS خالی است.

اکنون باید بتوانید <http://mysite.com:8000/account/login/> را در مرورگر خود باز کنید.

احراز هویت با استفاده از Facebook

به منظور اینکه کاربران بتوانند با حساب Facebook خود به سایت خود وارد شوند ، خط زیر را به تنظیم setting.py در پرونده AUTHENTICATION_BACKENDS اضافه کنید:

```
'social_core.backends.facebook.FacebookOAuth2',
```

برای افزودن تأیید اعتبار اجتماعی با فیس بوک ، به یک حساب کاربری توسعه دهنده فیس بوک و ایجاد یک برنامه جدید در فیس بوک نیاز دارید. <https://developers.facebook.com/apps/> را در مرورگر خود باز کنید.

عنوان زیر را در سایت مشاهده خواهید کرد:



روی دکمه افزودن یک برنامه جدید کلیک کنید. برای ایجاد شناسه برنامه جدید فرم زیر را مشاهده خواهید کرد:

Create a New App ID

Get started integrating Facebook into your app or website

Display Name

Bookmarks

Contact Email

antonio.mele@zenxit.com

By proceeding, you agree to the [Facebook Platform Policies](#)

[Cancel](#)

[Create App ID](#)

نشانک ها را به عنوان نام نمایش وارد کنید ، آدرس ایمیل مخاطب را اضافه کنید و روی ایجاد برنامه شناسه کلیک کنید.
داشبورد را برای برنامه جدید خود مشاهده خواهید کرد که ویژگی های مختلفی را که می توانید برای برنامه خود تنظیم کنید
نمایش می دهد. به دنبال کادر ورود به سیستم در Facebook Set-Up بروید و بر روی Create App ID کلیک کنید:



Facebook Login

The world's number one social login product.

[Read Docs](#)

[Set Up](#)

از شما خواسته می شود سیستم عامل را به شرح زیر انتخاب کنید:



Select the Web platform. You will see the following form:

1. Tell Us about Your Website

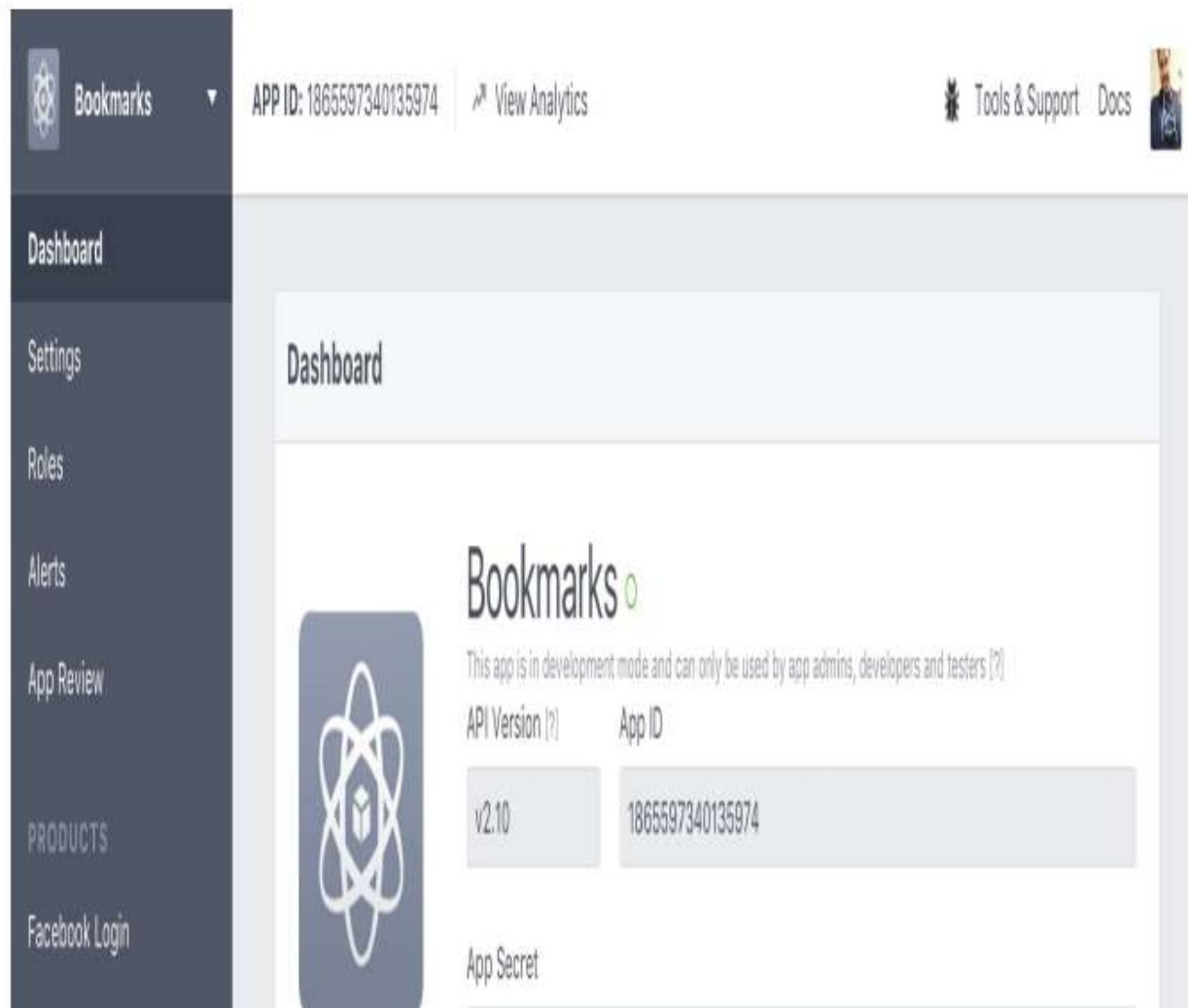
Tell us what the URL of your site is.

Site URL

Save

Continue

به عنوان آدرس سایت خود `http://mysite.com:8000/` را وارد کنید و بر روی دکمه ذخیره کلیک کنید. می توانید بقیه مراحل Quickstart را پرشن کنید. در منوی lefthand روی Dashboard کلیک کنید. چیزی شبیه به موارد زیر را مشاهده خواهید کرد:



کلیدهای appid و برنامه appSecret را کپی کنید و آنها را به پرونده زیر اضافه کنید:

```
SOCIAL_AUTH_FACEBOOK_KEY = 'XXX' # Facebook App ID  
SOCIAL_AUTH_FACEBOOK_SECRET = 'XXX' # Facebook App Secret
```

به صورت اختیاری ، می توانید یک تنظیمات SOCIAL_AUTH_FACEBOOK_SCOPE را با مجوزهای اضافی که می خواهید از کاربران فیس بوک برسید تعریف کنید:

```
SOCIAL_AUTH_FACEBOOK_SCOPE = ['email']
```

حال به بازگردید و بر روی Facebook Settings کلیک کنید. یک فرم با تنظیمات متعدد برای برنامه خود مشاهده خواهد کرد. در دامنه برنامه به شرح زیر اضافه کنید: mysite.com

App Domains

mysite.com 

روی ذخیره تغییرات کلیک کنید. سپس ، در منوی سمت چپ ، روی Facebook Login کلیک کنید. اطمینان حاصل کنید که فقط تنظیمات زیر فعال هستند:

ورود Client OAuth

ورود به وب OAuth

مرونگر جاسازی شده OAuth ورود

وارد /http://mysite.com:8000/social-auth/complete/facebook را در زیر URI های تغییر مسیر معترض کنید. انتخاب باید به این شکل باشد:

Facebook OAuth Settings



Client OAuth Login

Enables the standard OAuth client token flow. Secure your application and prevent abuse by locking down which token redirect URIs are allowed with the options below. Disable globally if not used. [?]



Web OAuth Login

Enables web based OAuth client login for building custom login flows. [?]



Force Web OAuth Reauthentication

When on, prompts people to enter their Facebook password in order to log in on the web. [?]



Embedded Browser OAuth Login

Enables browser control redirect URI for OAuth client login. [?]



Use Strict Mode for Redirect URIs

Only allow redirects that use the Facebook SDK or that exactly match the Valid OAuth Redirect URLs. Strongly recommended. [?]

Valid OAuth redirect URLs

<http://mysite.com:8000/social-auth/complete/facebook/>



Login from Devices

Enables the OAuth client login flow for devices like a smart TV. [?]

الگوی ثبت نام / login.html برنامه حساب خود را باز کنید و کد زیر را در انتهای بلوک محتوا اضافه کنید:

```
<div class="social">
  <ul>
    <li class="facebook"><a href="{% url "social:begin" "facebook" %}">Sign
      in with Facebook</a></li>
  </ul>
</div>
```

را در مرورگر خود باز کنید. اگر این صفحه ورود به شرح زیر است:

Log-in

Please, use the following form to log-in. If you don't have an account [register here](#)

Username:

[Sign in with Facebook](#)

Password:

[LOG-IN](#)[Forgotten your password?](#)

با کلیک بر روی دکمه ورود به سیستم با فیس بوک. شما به فیس بوک هدایت می شوید و یک گفتگوی معین را مشاهده می کنید که درخواست اجازه شما را برای دسترسی به برنامه نشانک ها به پروفایل عمومی فیس بوک شما می دهد:



Bookmarks will receive:
your public profile and email address. ⓘ

Edit This

Continue as Antonio

بر روی دکمه ... Continue as ... کلیک کنید. شما وارد صفحه داشبورد سایت خود خواهید شد. به یاد داشته باشید که ما این URL را در تنظیمات LOGIN_REDIRECT_URL تنظیم کرده ایم. همانطور که مشاهده می کنید ، اضافه کردن احراز هویت اجتماعی به سایت شما بسیار ساده است.

تأیید اعتبار با استفاده از توییتر

برای احراز هویت اجتماعی با استفاده از توییتر ، خط زیر را به تنظیم AUTHENTICATION_BACKENDS در پرونده setting.py پروژه خود اضافه کنید:

```
'social_core.backends.twitter.TwitterOAuth',
```

شما نیاز به ایجاد یک برنامه جدید در حساب توییتر خود دارید.

: https://apps.twitter.com/app/new را در مرورگر خود باز کنید. فرم زیر را مشاهده خواهید کرد:

Description *

Test Django application.

Your application description, which will be shown in user-facing authorization screens. Between 10 and 300 characters max.

Website *

http://mysite.com:8000/

Your application's publicly accessible home page, where users can go to download, make use of, or find out more information about your application. This fully-qualified URL is used in the source attribution for tweets created by your application and will be shown in user-facing authorization screens.

(If you don't have a URL yet, just put a placeholder here but remember to change it later.)

Callback URL

http://mysite.com:8000/social-auth/complete/twitter/

Where should we return after successfully authenticating? OAuth 1.0a applications should explicitly specify their `oauth_callback_url` on the request token step, regardless of the value given here. To restrict your application from using callbacks, leave this field blank.

جزئیات برنامه خود را از جمله تنظیمات زیر وارد کنید:

وب سایت: /http://mysite.com:8000

URL پاسخ به تماس: /http://mysite.com:8000/social-auth/complete/twitter

سپس بر روی ایجاد برنامه توییتر خود کلیک کنید. جزئیات برنامه را مشاهده خواهید کرد. روی Keys and Access کلیک کنید. شما باید اطلاعات زیر را مشاهده کنید: Tokens

Bookmarks

Details Settings

Keys and Access Tokens

Permissions

Application Settings

Keep the "Consumer Secret" a secret. This key should never be human-readable in your application.

Consumer Key (API Key) eJJU1AzzEQFJ6PAgqLjc18TH1

Consumer Secret (API
Secret) *****

Access Level Read and write ([modify app permissions](#))

کلیدهای Consumer Secret و Consumer Key را در تنظیمات زیر در پرونده settings.py پروژه خود کپی کنید:

```
SOCIAL_AUTH_TWITTER_KEY = 'XXX' # Twitter Consumer Key
SOCIAL_AUTH_TWITTER_SECRET = 'XXX' # Twitter Consumer Secret
```

حالا الگوی registration/login.html را ویرایش کنید و کد زیر را به عنصر اضافه کنید:

```
<li class="twitter"><a href="{% url "social:begin" "twitter" %}">Login with Twitter</a></li>
```

توییتر هدایت می شوید و از شما می خواهد که برنامه را به شرح زیر مجاز کنید:

Authorize Bookmarks Test to use your account?



Authorize app

Cancel

Bookmarks

mysite.com:8000/

Test Django application.

This application will be able to:

- Read Tweets from your timeline.
- See who you follow.

Will not be able to:

- Follow new people.
- Update your profile.
- Post Tweets for you.
- Access your direct messages.
- See your email address.
- See your Twitter password.

بر روی برنامه تأیید کلیک کنید. شما وارد صفحه داشبورد سایت خود خواهید شد.

تأیید اعتبار با استفاده از Google

احراز هویت OAuth2 Google را ارائه می دهد. می توانید درباره اجرای OAuth2 در <https://developers.google.com/identity/protocols/OAuth2> مطالعه کنید.

برای احراز هویت با استفاده از Google ، خط زیر را به تنظیمات AUTHENTICATION_BACKENDS در پرونده تنظیمات. نوع پروژه خود اضافه کنید:

```
'social_core.backends.google.GoogleOAuth2',
```

ابتدا باید یک کلید API در کنسول توسعه دهنده Google خود ایجاد کنید. را در مرورگر خود باز کنید. روی انتخاب یک پروژه کلیک کنید و یک پروژه جدید ایجاد کنید ، به شرح زیر:



New Project

 You have 12 projects remaining in your quota. [Learn more.](#)

Project name 

Bookmarks

Your project ID will be bookmarks-185117  [Edit](#)

[Create](#)

[Cancel](#)

پس از ایجاد پروژه ، تحت اعتبارنامه ، روی ایجاد اعتبارنامه کلیک کرده و شناسه مشتری OAuth را به شرح زیر انتخاب کنید:

APIs

Credentials

You need credentials to access APIs. [Enable the APIs that you plan to use](#) and then create the credentials that they require.

Depending on the API, you need an API key, a service account or an OAuth 2.0 client ID. [Refer to the API documentation](#) for details.

Create credentials ▾

API key

Identifies your project using a simple API key to check quota and access.

OAuth client ID

Requests user consent so your app can access the user's data.

Service account key

Enables server-to-server, app-level authentication using robot accounts.

Help me choose

Asks a few questions to help you decide which type of credential to use

از شما خواهد خواست که ابتدا صفحه رضایت را پیکربندی کنید:



To create an OAuth client ID, you must first set a product name on the consent screen.

[Configure consent screen](#)

صفحه قبل همان صفحه ای است که به کاربران نشان می دهد رضایت خود را برای دسترسی به سایت شما با حساب Google خود نشان دهند. روی دکمه صفحه پیکربندی رضایت کلیک کنید. آدرس ایمیل خود را انتخاب کنید، نشانک ها را تحت عنوان Product وارد کرده و روی دکمه Save کلیک کنید.

صفحه رضایت برنامه شما پیکربندی می شود، و به منظور هدایت شناسه مشتری خود هدایت می شوید.

فرم زیر را با اطلاعات زیر پر کنید:

نوع برنامه: برنامه وب را انتخاب کنید

نام: نشانک ها را وارد کنید

URI های هدایت مجدد مجاز: `/http://mysite.com:8000/socialauth/complete/google-oauth2`

فرم باید به صورت زیر باشد:

Application type

- Web application
- Android [Learn more](#)
- Chrome App [Learn more](#)
- iOS [Learn more](#)
- PlayStation 4
- Other

Name

Bookmarks

Restrictions

Enter JavaScript origins, redirect URIs or both

Authorised JavaScript origins

For use with requests from a browser. This is the origin URI of the client application. It cannot contain a wildcard (`https://*.example.com`) or a path (`https://example.com/subdir`). If you're using a non-standard port, you must include it in the origin URI.

`https://www.example.com`

Authorised redirect URIs

For use with requests from a web server. This is the path in your application that users are redirected to after they have authenticated with Google. The path will be appended with the authorisation code for access. Must have a protocol. Cannot contain URL fragments or relative paths. Cannot be a public IP address.

`http://mysite.com:8000/social-auth/complete/google-oauth2/`

X

`https://www.example.com/oauth2callback`

Create

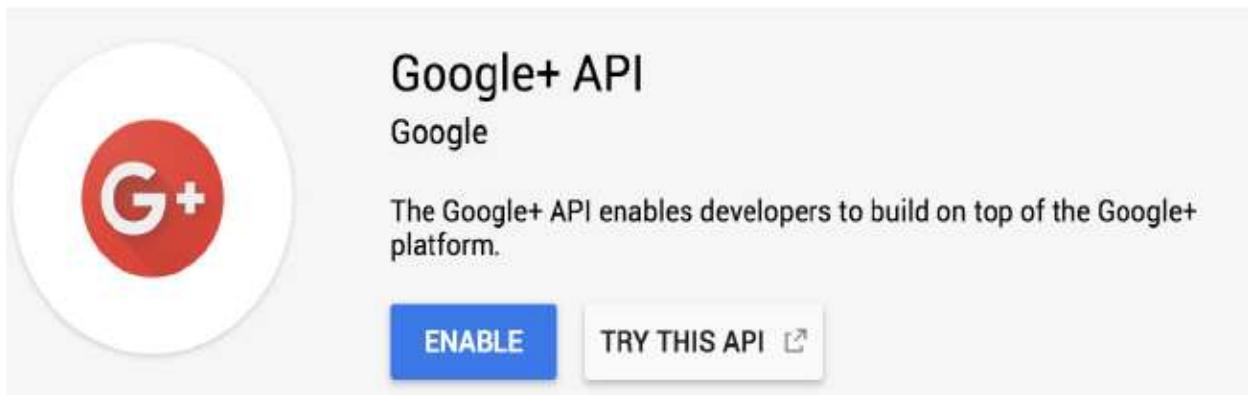
Cancel

بر روی دکمه ایجاد کلیک کنید. کلیدهای Client ID و ClientSecret را دریافت خواهید کرد. آنها را به پرونده تنظیمات خود

اضافه کنید ، مانند این:

```
SOCIAL_AUTH_GOOGLE_OAUTH2_KEY = 'XXX' # Google Consumer Key  
SOCIAL_AUTH_GOOGLE_OAUTH2_SECRET = 'XXX' # Google Consumer Secret
```

در منوی سمت چپ کنسول توسعه دهندگان Google ، در زیر بخش API و خدمات ، روی پیوند کتابخانه کلیک کنید. لیستی را مشاهده خواهید کرد که شامل تمام API های Google است. بر روی Google+ API کلیک کرده و سپس در صفحه زیر بر روی دکمه ENABLE کلیک کنید:



قالب login.html را ویرایش کنید و کد زیر را به عنصر <a> اضافه کنید:

```
<li class="google"><a href="{% url "social:begin" "google-oauth2" %}">Login  
with Google</a></li>
```

Log-in

Please, use the following form to log-in. If you don't have an account [register here](#)

Username:

[Sign in with Facebook](#)

Password:

[Login with Twitter](#)

LOG-IN

[Login with Google](#)

بر روی دکمه ورود با Google کلیک کنید. وارد سیستم شده و به صفحه داشبورد وب سایت خود هدایت می شوید.

شما احراز هویت اجتماعی را به پروژه خود اضافه کرده اید. با استفاده از Python Social Auth ، می توانید احراز هویت اجتماعی را با سایر خدمات آنلاین محبوب به راحتی پیاده سازی کنید.

خلاصه

در این فصل ، شما یاد گرفتید که چگونه یک سیستم تأیید هویت را در سایت خود ایجاد کنید و پروفایلهای کاربر سفارشی ایجاد کنید. همچنین احراز هویت اجتماعی را به سایت خود اضافه کردید.

در فصل بعد ، شما می آموزید که چگونه یک سیستم بوک مارک تصویر ایجاد کنید ، ریز عکسها را بسازید ، و نماهای AJAX بسازید.

فصل پنجم

اشتراک گذاری

مطالب در وب سایت

در فصل قبل ، شما ثبت نام کاربر و تأیید اعتبار را در وب سایت خود ایجاد کرده اید. شما آموخته اید که چگونه یک مدل پروفایل سفارشی برای کاربران خود ایجاد کنید و تأیید اعتبار اجتماعی را با شبکه های اصلی اجتماعی به سایت خود اضافه کنید.

در این فصل یاد می گیرید که چگونه یک نشانک جاوا اسکریپت ایجاد کنید تا محتوای سایر سایت ها را در وب سایت خود به اشتراک بگذارید و ویژگی های AJAX را با استفاده از jQuery و Django در پروژه خود پیاده سازی خواهید کرد.

این فصل نکات زیر را در بر می گیرد:

ایجاد روابط بسیار زیاد

رفتار شخصی سازی برای فرم ها

استفاده از jQuery با جنگو

ساخت یک نشانک jQuery

تولید تصاویر کوچک با استفاده از تصاویر کوچک

اجرای نماهای AJAX و ادغام آنها درجی کوئری

ایجاد دکوراتورهای سفارشی برای نمایش

ساخت صفحه بندی AJAX

ایجاد یک وب سایت نشانک تصویر

ما به کاربران این امکان را می دهیم که تصاویر خود را در وب سایت های دیگر و در سایت ما به اشتراک بگذارند. برای این کار،
ما باید کارهای زیر را انجام دهیم:

1. برای ذخیره تصاویر و اطلاعات آنها مدلی را تعریف کنید

2. یک فرم و یک نمای برای ایجاد بارگذاری تصاویر ایجاد کنید

3. سیستمی بسازید تا کاربران بتوانند تصاویری را که در سایتها خارجی پیدا می کنند ارسال کنند

ابتدا با دستور زیر یک برنامه جدید در فهرست پروژه های نشانک خود ایجاد کنید:

```
django-admin startapp images
```

برنامه جدید را در پرونده setting.py به تنظیمات INSTALLED_APPS اضافه کنید:

```
INSTALLED_APPS = [  
    # ...  
    'images.apps.ImagesConfig',  
]
```

ما برنامه تصاویر را در پروژه فعال کرده ایم.

ساختن مدل تصویر

پرونده model.py برنامه تصاویر را ویرایش کنید و کد زیر را به آن اضافه کنید:

```
from django.db import models
from django.conf import settings

class Image(models.Model):
    user = models.ForeignKey(settings.AUTH_USER_MODEL,
                            related_name='images_created',
                            on_delete=models.CASCADE)
    title = models.CharField(max_length=200)
    slug = models.SlugField(max_length=200,
                           blank=True)
    url = models.URLField()
    image = models.ImageField(upload_to='images/%Y/%m/%d/')
    description = models.TextField(blank=True)
    created = models.DateTimeField(auto_now_add=True,
                                   db_index=True)

    def __str__(self):
        return self.title
```

این مدلی است که ما برای ذخیره تصاویر بوک مارک شده از سایتها مختلف استفاده خواهیم کرد. باید نگاهی به زمینه های این مدل بیاندازیم:

user: این موضوع کاربر را نشان می دهد که این تصویر را علامت گذاری کرده است. این یک زمینه مهم خارجی است زیرا یک رابطه یک به بسیاری را مشخص می کند. یک کاربر می تواند چندین تصویر ارسال کند، اما هر تصویر توسط یک کاربر واحد ارسال می شود. ما برای پارامتر on_delete از CASCADE استفاده می کنیم تا هنگام حذف کاربر، تصاویر مرتبط نیز حذف شوند.

عنوان: عنوانی برای تصویر.

slug: برچسب کوتاهی است که فقط شامل حروف، اعداد، تأکیدات زیر یا شکلها است که برای ساختن URL های دوست داشتنی برای SEO استفاده می شود.

URL: URL اصلی برای این تصویر.

تصویر: پرونده تصویر.

توضیحات: توضیحی اختیاری برای تصویر.

ایجاد شده: تاریخ و زمانی که زمان ایجاد شی در پایگاه داده را نشان می دهد. از آنجاکه ما از `auto_now_add` استفاده می کنیم ، این `DateTime` هنگام ایجاد شی به طور خودکار تنظیم می شود. ما از `db_index = True` درست استفاده می کنیم تا Django برای این قسمت یک فهرست در بانک اطلاعاتی ایجاد کند.

ما روش `save()` از مدل تصویر را نادیده می گیریم تا بطور خودکار میدان `Slug` را بر اساس مقدار قسمت عنوان تولید کنیم. تابع `slugify()` را وارد کنید و یک روش `save()` را به مدل تصویر اضافه کنید ، به شرح زیر:

```
from django.utils.text import slugify

class Image(models.Model):
    # ...
    def save(self, *args, **kwargs):
        if not self.slug:
            self.slug = slugify(self.title)
        super(Image, self).save(*args, **kwargs)
```

در کد قبلی ، ما از تابع `slugify()` ارائه شده توسط Django استفاده می کنیم تا در صورت تهیه هیچ خزنده ، به طور خودکار از حلقوی تصویر برای عنوان داده شده استفاده نشود. سپس ، شی را ذخیره می کنیم. ما برای تصاویر به طور خودکار `slug` را تولید می کنیم تا کاربران مجبور نباشند به صورت دستی برای هر تصویر یک `slug` وارد کنند.

Creating many-to-many relationships

ما یک قسمت دیگر به مدل `Image` اضافه می کنیم تا کاربرانی که دوست دارند تصویری را ذخیره کنند. در این حالت به یک رابطه بسیار زیاد نیاز خواهید داشت زیرا ممکن است کاربر چند تصویر را دوست داشته باشد و هر تصویر را می توان توسط چندین کاربر دوست داشت.

قسمت زیر را به مدل `Image` اضافه کنید:

```
users_like = models.ManyToManyField(settings.AUTH_USER_MODEL,
                                    related_name='images_liked',
                                    blank=True)
```

هنگامی که شما یک ManyToManyField را تعریف می کنید ، Django با استفاده از کلیدهای اصلی هر دو مدل یک میز پیوست واسطه ایجاد می کند. ManyToManyField را می توان در هر یک از دو مدل مرتبط تعریف کرد.

همانطور که در زمینه های ForeignKey ، ویژگی ManyToManyField از related_name به ما اجازه می دهد تا رابطه را از موضوع مرتبط به این مورد نامگذاری کنیم. زمینه های ManyToManyField یک مدیر بسیار زیاد را در اختیار شما قرار می دهد که به ما امکان می دهد اشیاء مرتبط مانند تصویر.all() یا از یک شیء کاربر مانند user.images_liked.all را بازیابی کنیم.

خط فرمان را باز کنید و دستور زیر را برای ایجاد یک مهاجرت اولیه اجرا کنید:

```
python manage.py makemigrations images
```

باید خروجی زیر را مشاهده کنید:

```
Migrations for 'images':
  images/migrations/0001_initial.py
    - Create model Image
```

اکنون ، دستور زیر را برای اعمال مهاجرت خود اجرا کنید:

```
python manage.py migrate images
```

شما خروجی دریافت خواهید کرد که شامل خط زیر است:

```
Applying images.0001_initial... OK
```

مدل Image اکنون در پایگاه داده همگام سازی شده است.

ثبت مدل تصویر در سایت مدیریت

فایل admin.py برنامه تصاویر را ویرایش کنید و مدل تصویر را در سایت مدیریت ثبت کنید ، به شرح زیر:

```
from django.contrib import admin
from .models import Image

@admin.register(Image)
class ImageAdmin(admin.ModelAdmin):
    list_display = ['title', 'slug', 'image', 'created']
    list_filter = ['created']
```

سرور توسعه را با Python manage.py شروع کنید

دستور /runserver را در مرورگر خود باز کنید ، و مدل تصویر را در سایت مدیریت مشاهده خواهید کرد ، مانند این:

IMAGES

Images

 Add  Change

ارسال مطالب از وب سایت های دیگر

ما به کاربران این امکان را می دهیم که تصاویر را از وب سایت های خارجی نشانک گذاری کنند.

کاربر آدرس URL تصویر ، عنوان و توضیحات اختیاری را ارائه می دهد. برنامه ما تصویر را بارگیری می کند و یک شی تصویر جدید را در پایگاه داده ایجاد می کند. بباید با ساختن یک فرم برای ارسال تصاویر جدید شروع کنیم. یک پرونده جدید را در فهرست برنامه تصاویر ایجاد کنید و کد زیر را به آن اضافه کنید:

```
from django import forms
from .models import Image

class ImageCreateForm(forms.ModelForm):
    class Meta:
        model = Image
        fields = ('title', 'url', 'description')
        widgets = {
            'url': forms.HiddenInput,
        }
```

همانطور که در این کد قبلی مشاهده می کنید ، این فرم یک فرم Image است که از مدل ModelForm ساخته شده است ، و فقط شامل عنوان ، URL و قسمت های توضیحات است. کاربران آدرس اینترنتی تصویر را مستقیماً در قالب وارد نمی کنند. در عوض ، ما برای انتخاب تصویری از سایت خارجی ، به آنها ابزار جاوا اسکریپت ارائه می دهیم و فرم ما URL آن را به عنوان پارامتر دریافت می کند. ما برای استفاده از ویجت HiddenInput از ویجت پیش فرض قسمت URL رد می کنیم. این ویجت به عنوان یک عنصر ورودی HTML با یک type="hidden" ارائه می شود. ما از این ویجت استفاده می کنیم زیرا نمی خواهیم این قسمت برای کاربران قابل مشاهده باشد.

تمیز کردن قسمت های فرم برای تأیید صحت URL تصویر ارائه شده ، بررسی خواهیم کرد که نام پرونده با پسوند jpg یا jpeg به پایان می رسد تا فقط اجازه دسترسی به پرونده های JPEG را داشته باشد. همانطور که در فصل قبل دیدید ، <fieldname> به شما امکان می دهد روش های فرم را برای تمیز کردن زمینه های خاص با استفاده از نماد تمیز _() تعريف کنید.

در صورت وجود ، هنگامی که با is_valid() به عنوان نمونه تماس می گیرید ، این روش برای هر قسمت اجرا می شود. در روش تمیز ، می توانید مقدار این فیلد را تغییر داده و یا در صورت نیاز هرگونه خطای اعتبار سنجی را برای این قسمت بالا ببرید. روش زیر را به ImageCreateForm اضافه کنید:

```
def clean_url(self):
    url = self.cleaned_data['url']
    valid_extensions = ['jpg', 'jpeg']
    extension = url.rsplit('.', 1)[1].lower()
    if extension not in valid_extensions:
        raise forms.ValidationError('The given URL does not '
                                    'match valid image extensions.')
    return url
```

در کد قبلی ، ما برای تمیز کردن قسمت URL ، روشی Clean_url() را تعريف می کنیم. کد به شرح زیر عمل می کند:

1. ما با دستیابی به دیکشنری cleaned_data نمونه فرم ، مقدار زمینه URL را بدست می آوریم.
2. برای دریافت پسوند پرونده ، URL را تقسیم می کنیم و بررسی می کنیم که آیا این یکی از پسوندهای معتبر است یا خیر. اگر پسوند نامعتبر باشد ، ما ValidationError را بالا می بریم و نمونه فرم اعتبار نخواهد یافت. در اینجا ، ما یک اعتبار سنجی بسیار ساده را انجام می دهیم.

می توانید از روش‌های پیشرفته تری برای بررسی اینکه آیا URL داده شده یک فایل تصویری معتبر را ارائه می دهد استفاده کنید.

علاوه بر تأیید اعتبار URL مشخص شده ، ما نیز نیاز به بارگیری فایل تصویری و ذخیره آن خواهیم داشت. برای مثال می توانیم از نمایی که فرم را در اختیار دارد استفاده کنیم تا پرونده تصویر بارگیری شود. در عوض ، ما با غلبه بر روش save() فرم مدل خود ، یک روش کلی را برای انجام این کار هر بار که فرم ذخیره شود ، انجام خواهیم داد.

نادیده گرفتن روش save() در ModelForm

همانطور که می دانید ModelForm یک روش save را برای ذخیره نمونه مدل فعلی در پایگاه داده و بازگرداندن شی فراهم می کند. این روش یک پارامتر boolean commit را دریافت می کند ، که به شما این امکان را می دهد تا مشخص کنید که آیا شیء باید به پایگاه داده باقی بماند یا خیر. اگر متعهد False باشد ، روش save() نمونه ای را برای گرداند اما آن را به پایگاه داده ذخیره نمی کند. ما برای بازیابی تصویر داده شده و ذخیره آن ، روش save() فرم خود را نادیده می گیریم.

واردات زیر را در بالای پرونده فرم.py اضافه کنید:

```
from urllib import request
from django.core.files.base import ContentFile
from django.utils.text import slugify
```

سپس () زیر را به فرم ImageCreateForm اضافه کنید:

```

def save(self, force_insert=False,
        force_update=False,
        commit=True):
    image = super(ImageCreateForm, self).save(commit=False)
    image_url = self.cleaned_data['url']
    image_name = '{}.{}'.format(slugify(image.title),
                                image_url.rsplit('.', 1)[1].lower())

    # download image from the given URL
    response = request.urlopen(image_url)
    image.image.save(image_name,
                     ContentFile(response.read()),
                     save=False)
    if commit:
        image.save()
    return image

```

ما `save()` را نادیده می‌گیریم و پارامترهای مورد نیاز `ModelForm` را نگه می‌داریم. کد قبلی به شرح زیر توضیح داده شده است:

1. با فراخوانی متده `save()` فرم با `commit=True` ، نمونه جدیدی ایجاد می‌کنیم.
2. ما URL را از فرهنگ لغت `cleaned_data` فرم دریافت می‌کنیم.
3. ما با ترکیب اسلای عنوان با پسوند اصلی پرونده ، نام تصویر را تولید می‌کنیم.
- 4- ما از ماثول Python `urllib` برای بارگیری تصویر استفاده می‌کنیم و سپس متده `save()` را از قسمت تصویر صدا می‌نامیم ، و آن را به یک شی `ContentFile` می‌رسانیم که با محتواهی پرونده بارگیری شده فوری می‌شود. به این روش ، ما فایل را در فهرست رسانه ای پروژه خود ذخیره می‌کنیم. ما همچنین از پارامتر `save=False` برای جلوگیری از ذخیره شی در پایگاه داده ، عبور می‌کنیم.
- 5- برای حفظ همان رفتار با روش `save()` که ما آن را رد می‌کنیم ، فقط در صورت درست بودن پارامتر تعهد ، فرم را در بانک اطلاعات ذخیره می‌کنیم.

در حال حاضر ، برای رسیدگی به فرم به یک نمای نیاز خواهیم داشت. پرونده views.py برنامه تصاویر را ویرایش کنید و کد زیر را به آن اضافه کنید:

```
from django.shortcuts import render, redirect
from django.contrib.auth.decorators import login_required
from django.contrib import messages
from .forms import ImageCreateForm

@login_required
def image_create(request):
    if request.method == 'POST':
        # form is sent
        form = ImageCreateForm(data=request.POST)
        if form.is_valid():
            # form data is valid
            cd = form.cleaned_data
            new_item = form.save(commit=False)

            # assign current user to the item
            new_item.user = request.user
            new_item.save()
            messages.success(request, 'Image added successfully')

            # redirect to new created item detail view
            return redirect(new_item.get_absolute_url())
    else:
        # build form with data provided by the bookmarklet via GET
        form = ImageCreateForm(data=request.GET)

    return render(request,
                  'images/image/create.html',
                  {'section': 'images',
                   'form': form})
```

برای جلوگیری از دسترسی کاربران غیرمجاز ، یک دکوراتور login_required را به نمای image_create اضافه می کنیم .
اینگونه عمل می کند:

1. ما از داده های اولیه از طریق GET انتظار داریم تا نمونه ای از فرم ایجاد شود. این داده ها شامل ویژگی های url و عنوان یک تصویر از وب سایت خارجی است و از طریق JavaScript توسط ابزار که بعداً ایجاد خواهیم کرد ارائه می شود. در حال حاضر ، ما فقط فرض می کنیم که این داده ها در ابتدا در آنجا خواهند بود.

2. اگر فرم ارسال شد ، بررسی می کنیم که آیا صحت دارد یا خیر. اگر داده فرم معتبر است ، ما یک نمونه جدید تصویر ایجاد می کنیم ، اما مانع از آن نمی شود که شیء در پایگاه داده ذخیره شود ، اما با این کار از commit=False به روش save() فرم جلوگیری می کند.

3. کاربر فعلی را به شیء تصویر جدید اختصاص می دهیم. اینگونه می توانیم بدانیم چه کسی هر تصویر را باگذاری کرده است.

4- شی تصویر را در پایگاه داده ذخیره می کنیم.

5- در آخر ، ما با استفاده از چارچوب پیام رسانی Django یک پیام موفقیت آمیز ایجاد می کنیم و کاربر را به URL متعارف تصویر جدید هدایت می کنیم. ما هنوز روش get_absolute_url() از مدل تصویر را پیاده سازی نکرده ایم. بعداً این کار را خواهیم کرد

یک پرونده urls.py جدید را درون برنامه تصاویر ایجاد کنید و کد زیر را به آن اضافه کنید:

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

app_name = 'images'

urlpatterns = [
    path('create/', views.image_create, name='create'),
]
```

فایل اصلی urls.py پروژه بوکمارک ها را ویرایش کنید تا الگوهای مربوط به برنامه تصاویر را درج کنید ، به شرح زیر:

```
urlpatterns = [
    path('admin/', admin.site.urls),
    path('account/', include('account.urls')),
    path('social-auth/',
        include('social_django.urls', namespace='social')),
    path('images/', include('images.urls', namespace='images')),
]
```

سرانجام ، شما نیاز به ایجاد یک الگوی برای ارائه فرم دارید.

ساختر فهرست زیر را در فهرست برنامه کاربردی تصاویر ایجاد کنید:

```
templates/
    images/
        image/
            create.html
```

الگوی جدید create.html را ویرایش کنید و کد زیر را به آن اضافه کنید:

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

{% block title %}Bookmark an image{% endblock %}

{% block content %}
<h1>Bookmark an image</h1>

```

```
<form action"." method="post">
{{ form.as_p }}
{% csrf_token %}
<input type="submit" value="Bookmark it!">
</form>
{% endblock %}
```

اکنون ، در مرورگر خود ، شامل یک عنوان و پارامترهای GET URL و URL پارامترهای GET را در مرورگر خود / را باز کنید ، و یک URL تصویری JPEG موجود را در دوی ارائه دهید.

به عنوان مثال ، می توانید از URL زیر استفاده کنید:

title=%20Django%20and%20Duke&url=http://upload.wikimedia.org/wikipedia/commons/8/85/Django_Reinhardt_and_Duke_Ellington_%28Gottlieb%29.j
.pg

فرم را با پیش نمایش تصویر خواهید دید ، مانند شکل زیر:

Bookmark an image



Title:

Django and Duke

Description:

BOOKMARK IT!

توضیحی را اضافه کنید و روی BOOKMARK IT کلیک کنید! سخنان جدید تصویر در پایگاه داده شما ذخیره می شود. با این وجود خطای دریافت خواهید کرد که نشان می دهد که مدل Image روش get_absolute_url را به شرح زیر ندارد:

AttributeError at /images/create/

'Image' object has no attribute 'get_absolute_url'

اکنون نگران این موضوع نباشید. ما می خواهیم این روش را بعداً اضافه کنیم.
را در مرورگر خود باز کنید و تأیید کنید که شیء تصویر جدید ذخیره شده است ، مانند

Action: ----- 0 of 1 selected

<input type="checkbox"/>	TITLE	SLUG	IMAGE	CREATED
<input type="checkbox"/>	Django and Duke	django-and-duke	images/2017/11/05/django-and-duke.jpg	Dec. 16, 2017

ساخت یک نشانک با jQuery

نشانک یک نشانه ذخیره شده در یک مرورگر وب است که شامل کد JavaScript برای گسترش عملکرد مرورگر است. هنگامی که روی نشانک کلیک می کنید ، کد JavaScript در وب سایت نمایش داده شده در مرورگر اجرا می شود. این برای ساختن ابزارهایی که با سایر وب سایتها در تعامل هستند بسیار مفید است.

برخی از خدمات آنلاین ، مانند Pinterest ، نشانک های خود را پیاده سازی می کنند تا به کاربران اجازه دهنند محتوای سایر سایتها را بر روی پلتفرم خود به اشتراک بگذارند. ما به یک روش مشابه ، یک بوکمارک ایجاد می کنیم تا به کاربران اجازه دهیم تصاویر را از سایر سایتها در وب سایت ما به اشتراک بگذارند.

برای ساخت نشانک خود از jQuery استفاده خواهیم کرد.jQuery یک فریم ورک محبوب JavaScript است که به شما امکان می دهد سریعتر عملکرد سمت مشتری را توسعه دهید. می توانید اطلاعات بیشتر در مورد jQuery را در وب سایت رسمی آن ، <https://jquery.com> بخوانید.

به این ترتیب کاربران شما یک نشانک را به مرورگر خود اضافه می کنند و از آن استفاده می کنند:

1. کاربر پیوندی را از سایت شما به نشانک های مرورگر خود می کشد. پیوند شامل کد JavaScript در ویژگی href آن است. این کد در بوکمارک ذخیره می شود.
2. کاربر به هر وب سایت هدایت می شود و روی بوک مارک کلیک می کند. کد JavaScript از بوک مارک اجرا شده است.

از آنجاکه کد JavaScript به عنوان یک نشانک ذخیره می شود ، نمی توانید بعداً آن را به روز کنید. این یک اشکال مهمی است که شما می توانید با پیاده سازی یک اسکریپت لانچر بارگذاری کنید تا نشانک اصلی JavaScript از یک URL بارگیری شود. کاربران شما این اسکریپت پرتاب را به عنوان یک نشانک ذخیره می کنند ، و شما قادر خواهید بود تا هر زمان که بخواهید ، کد نشانک را به روز کنید. این رویکردی است که ما برای ساخت نشانک خود استفاده خواهیم کرد. بیا شروع کنیم!

یک الگوی جدید در زیر تصاویر / قالب ها ایجاد کنید و آن را bookmarklet_launcher.js نام دهید. این اسکریپت لانچر خواهد بود. کد جاوا اسکریپت زیر را به این پرونده اضافه کنید:

```
(function(){
    if (window.myBookmarklet !== undefined){
        myBookmarklet();
    }
    else {

        document.body.appendChild(document.createElement('script')).src='http://127.0
.0.1:8000/static/js/bookmarklet.js?
r='+Math.floor(Math.random()*99999999999999999999);
    }
})();
```

اسکریپت قبلی با بررسی اینکه متغیر myBookmarklet تعریف شده است ، کشف می کند که نشانک قبلًا بارگذاری شده است یا خیر. با این کار ، اگر کاربر بارها و بارها روی نشانک کلیک کند ، از بارگذاری مجدد آن جلوگیری می کنیم. اگر myBookmarklet تعریف نشده باشد ، ما با اضافه کردن یک عنصر <script> به سند ، پرونده JavaScript دیگری را بارگذاری می کنیم.

برچسب اسکریپت اسکریپت bookmarklet.js را با استفاده از یک عدد تصادفی به عنوان یک پارامتر بارگذاری می کند تا از بارگذاری پرونده در حافظه نهان مرورگر جلوگیری کند.

کد نشانک واقعی در پرونده استاتیک bookmarklet.js اقامت خواهد داشت. به ما این امکان را می دهد تا بدون ایجاد نیاز به کاربران خود ، نشانک قبلی را که قبلًا به مرورگر خود اضافه کرده بودند ، کد نشانک خود را به روز کنیم. بیایید پرتاب نشانک را به صفحات داشبورد اضافه کنیم تا کاربران ما بتوانند آن را در نشانک های خود کپی کنند.

الگوی حساب / dashboard.html برنامه حساب را ویرایش کنید و به صورت زیر ظاهر کنید:

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

{% block title %}Dashboard{% endblock %}

{% block content %}
<h1>Dashboard</h1>

{% with total_images_created=request.user.images_created.count %}
<p>Welcome to your dashboard. You have bookmarked {{ total_images_created }} image{{ total_images_created|pluralize }}.</p>
{% endwith %}

<p>Drag the following button to your bookmarks toolbar to bookmark images from other websites – <a href="javascript:{% include "bookmarklet_launcher.js" %}" class="button">Bookmark it</a><p>
<p>You can also <a href="{% url "edit" %}">edit your profile</a> or <a href="{% url "password_change" %}">change your password</a>.<p>
{% endblock %}
```

داشبورد در حال حاضر تعداد کل تصاویر علامت گذاری شده توسط کاربر را نشان می دهد. ما از تگ قالب { % with % } استفاده می کنیم تا یک متغیر با تعداد کل تصاویر علامت گذاری شده توسط کاربر فعلی تنظیم کنیم. همچنین پیوندی با یک ویژگی href می کنیم که شامل اسکریپت پرتاب نشانک است. این کد JavaScript را از الگوی [drag](#) را درج می کند href "bookmarklet_launcher.js" در خواهیم کرد.

را در مرورگر خود باز کنید. باید صفحه زیر را مشاهده کنید:

Bookmarks

My dashboard Images People

Hello Antonio, Logout

Dashboard

Welcome to your dashboard. You have bookmarked 1 image.

Drag the following button to your bookmarks toolbar to bookmark images from other websites →

BOOKMARK IT

You can also [edit your profile](#) or [change your password](#).

اکنون دایرکتوری ها و پرونده های زیر را درون فهرست برنامه کاربردی تصاویر ایجاد کنید:

```
static/  
  js/  
    bookmarklet.js
```

شما می توانید یک static/css/ directory را در زیر فهرست برنامه های کاربردی تصاویر ، در کدی که همراه با این فصل است ، پیدا کنید. css / فهرست را در static/ directory کد خود کپی کنید. پرونده bookmarklet.css سبک های css / bookmarklet.js را ویرایش کنید و کد جاوا اسکریپت زیر را به آن اضافه کنید:

```

(function(){
    var jquery_version = '3.3.1';
    var site_url = 'http://127.0.0.1:8000/';
    var static_url = site_url + 'static/';
    var min_width = 100;
    var min_height = 100;

    function bookmarklet(msg) {
        // Here goes our bookmarklet code
    };

    // Check if jQuery is loaded
    if(typeof window.jQuery != 'undefined') {
        bookmarklet();
    } else {
        // Check for conflicts
        var conflict = typeof window.$ != 'undefined';
        // Create the script and point to Google API
        var script = document.createElement('script');
        script.src = '//ajax.googleapis.com/ajax/libs/jquery/' +
            jquery_version + '/jquery.min.js';
        // Add the script to the 'head' for processing
        document.head.appendChild(script);
        // Create a way to wait until script loading
        var attempts = 15;
        (function(){
            // Check again if jQuery is undefined
            if(typeof window.jQuery == 'undefined') {
                if(--attempts > 0) {
                    // Calls himself in a few milliseconds
                    window.setTimeout(arguments.callee, 250)
                } else {
                    // Too much attempts to load, send error
                    alert('An error occurred while loading jQuery')
                }
            } else {
                bookmarklet();
            }
        })();
    }
})()

```

این اسکریپت لودر اصلی jQuery است. اگر از قبل در وب سایت فعلی بارگیری شده باشد ، از استفاده از jQuery مراقبت می کند. اگر جی کوئری بارگیری نشده باشد ، اسکریپت jQuery را از شبکه تحويل محتوا Google ، که میزبان چارجوب های

معروف JavaScript است ، بارگیری می کند. هنگامی که جی کوئری بارگیری می شود ، عملکرد نشانک () را که شامل کد نشانک ما است ، انجام می دهد. ما همچنین برخی از متغیرها را در بالای پرونده تنظیم می کنیم:

و site_url:jQuery نسخه j برای بارگذاری static_url: URL پایه وب سایت و پایگاه ما

min_width و min_height : حداقل عرض و ارتفاع در پیکسل ها برای تصاویری که نشانک ما مسی می کند در سایت بباید اکنون بباید عملکرد نشانک را عملی کنیم. عملکرد() bookmarklet را ویرایش کنید تا به شکل زیر باشد:

```
function bookmarklet(msg) {
    // load CSS
    var css = jQuery('<link>');
    css.attr({
        rel: 'stylesheet',
        type: 'text/css',
        href: static_url + 'css/bookmarklet.css?r=' +
        Math.floor(Math.random()*9999999999999999)
    });
    jQuery('head').append(css);

    // load HTML
    box_html = '<div id="bookmarklet"><a href="#" id="close">&times;</a>
    <h1>Select an image to bookmark:</h1><div class="images"></div></div>';
    jQuery('body').append(box_html);

    // close event
    jQuery('#bookmarklet #close').click(function(){
        jQuery('#bookmarklet').remove();
    });
};
```

کد قبلی به شرح زیر عمل می کند:

1. ما شیوه نامه bookmarklet.css را با استفاده از یک عدد تصادفی به عنوان یک پارامتر بارگذاری می کنیم تا از برگشت مرورگر پرونده ذخیره شده جلوگیری شود.

2. ما HTML دلخواه را به عنصر `<body>` عنصر وب سایت فعلی اضافه می کنیم. این شامل یک عنصر `<div>` است که تصاویر موجود در وب سایت فعلی را در بر می گیرد.

3. ما رویدادی را اضافه می کنیم که HTML ما را از آن حذف می کند

هنگامی که کاربر روی پیوند نزدیک به بلاک HTML مالکیک می کند ، از انتخابگر `#close` استفاده می کنیم تا عنصر HTML را با شناسه ای به نام نزدیک پیدا کنیم ، که عنصر اصلی را با یک شناسه به نام نشانک دارد. انتخاب کنندگان jQuery به شما امکان می دهد عناصر HTML را پیدا کنید. یک انتخاب کننده jQuery تمام عناصر یافت شده توسط انتخاب کننده CSS داده شده را برمی گرداند. لیستی از انتخاب کنندگان jQuery را می توانید در <https://api.jquery.com/category/selectors> پیدا کنید.

پس از بارگذاری سبک های CSS و کد HTML برای بوکمارک ، باید تصاویر را در وب سایت پیدا کنیم. کد جاوا اسکریپت زیر را در پایین تابع `(function() { })` اضافه کنید:

```
// find images and display them
jQuery.each(jQuery('img[src$=".jpg"]'), function(index, image) {
  if (jQuery(image).width() >= min_width && jQuery(image).height()
    >= min_height)
  {
    image_url = jQuery(image).attr('src');
    jQuery('#bookmarklet .images').append('<a href="#"></a>');
  }
});
```

کد قبلی از انتخاب کننده `img [src $ = ".jpg"]` برای یافتن همه عناصر HTML استفاده می کند ، که ویژگی `src` با یک رشته `jpg` به پایان می رسد. این بدان معنی است که ما تمام تصاویر JPEG نمایش داده شده در وب سایت فعلی را جستجو خواهیم کرد. ما با استفاده از `jQuery.each()`، بیش از نتایج تکرار می کنیم. تصاویر را با سایز بزرگتر از اندازه مشخص شده با متغیرهای `min_width` و `min_height` به ظرف `<div class = "images">` می اضافه می کنیم.

شما باید بتوانید نشانک را در هر سایتی از جمله سایتهاي که از طریق HTTPS ارائه می شوند بارگیری کنید. SSL بسیار مورد استفاده قرار گرفته است و امروزه بیشتر وب سایت ها از طریق HTTPS به محتوا سرویس می دهند. به دلیل امنیتی ، مرورگر شما از اجرای دفترچه یادداشت بر روی HTTP در سایتی که از طریق HTTPS استفاده می شود جلوگیری می کند.

سرور توسعه Django فقط برای توسعه در نظر گرفته شده است و از HTTPS پشتیبانی نمی کند. برای آزمایش نشانک از طریق HTTPS ، از Ngrok استفاده خواهیم کرد. Ngrok ابزاری است که تونل ایجاد می کند تا بتواند localhost را از طریق HTTP و HTTPS در معرض اینترنت قرار دهد.

را برای سیستم عامل خود از <https://ngrok.com/download> بارگیری کنید و با استفاده از دستور زیر آن را از پوسته اجرا کنید:

```
./ngrok http 8000
```

با دستور قبلی ، به Ngrok می گویید که یک تونل را به localhost خود در پورت 8000 ایجاد کرده و یک نام میزبان قابل دسترسی به اینترنت برای آن اختصاص دهید. باید خروجی مشابه این مورد را مشاهده کنید:

Session	Status	online				
Version	2.2.8					
Region	United States (us)					
Web Interface	http://127.0.0.1:4040					
Forwarding	http://3f6ad53c.ngrok.io -> localhost:8000					
Forwarding	https://3f6ad53c.ngrok.io -> localhost:8000					
Connections	ttl	open	rt1	rt5	p50	p90
	0	0	0.00	0.00	0.00	0.00

به ما می گوید که سایت ما که با استفاده از سرور توسعه دهنده Django به صورت محلی در localhost درگاه 8000 اجرا می شود ، از طریق آدرس های <https://3f6ad53c.ngrok.io> و <http://3f6ad53c.ngrok.io> در اینترنت در

دسترسی فارمی گیرد. پروتکل های HTTP و HTTPS به ترتیب. Ngrok همچنین URL را برای دسترسی به یک رابط وب فراهم می کند که اطلاعات مربوط به درخواست های ارسال شده به سرور را در پورت 4040 نمایش می دهد.

پرونده settings.py چون خود را ویرایش کنید و میزبان ارائه شده توسط Ngrok را به تنظیمات ALLOWED_HOSTS اضافه کنید ، به شرح زیر:

```
ALLOWED_HOSTS = [
    'mysite.com',
    'localhost',
    '127.0.0.1',
    '3f6ad53c.ngrok.io'
]
```

این امر به شما امکان می دهد از طریق نام میزبان جدید برنامه را به خدمت بگیرید. سپس آدرس URL /https://3f6ad53c.ngrok.io/account/login را در مرورگر خود باز کنید و میزبان را جایگزین آن کنید که توسط Ngrok تهیه شده است. قادر خواهید بود سایت ورود را مشاهده کنید.

الگوی bookmarklet_launcher.js را ویرایش کنید و آدرس HTTPS ارائه شده توسط Ngrok جایگزین کنید ، به شرح زیر:

```
(function(){
    if (window.myBookmarklet !== undefined){
        myBookmarklet();
    }
    else {

        document.body.appendChild(document.createElement('script')).src='https://3f6a
d53c.ngrok.io/static/js/bookmarklet.js?
r='+Math.floor(Math.random()*99999999999999999999);
    }
})();
```

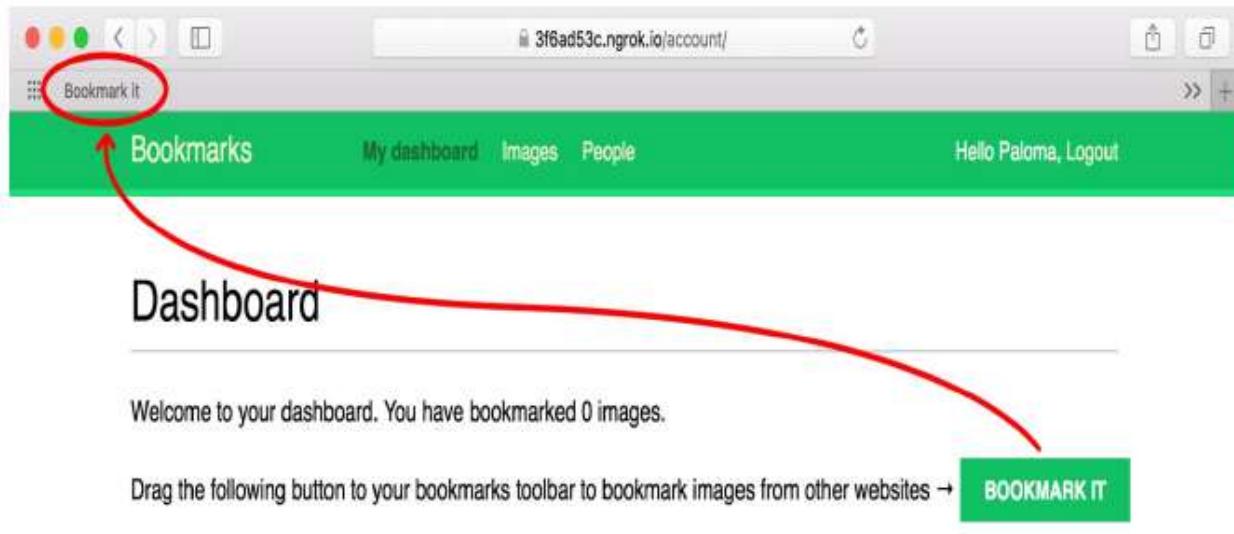
پرونده استاتیک / js را ویرایش کنید و به خط زیر نگاهی بیندازید:

```
var site_url = 'http://127.0.0.1:8000/';
```

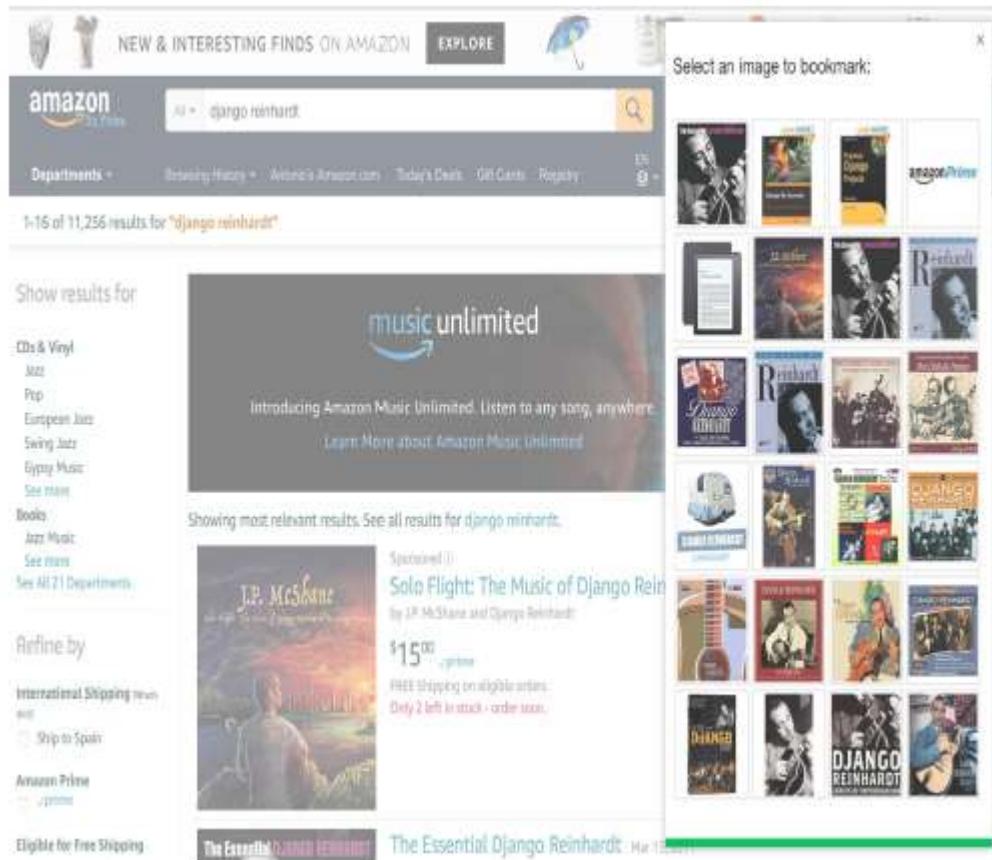
خط قبلی را با خط زیر جایگزین کنید ، از جمله URL HTTPS ارائه شده توسط Ngrok :

```
var site_url = 'https://3f6ad53c.ngrok.io/';
```

Ngrok / را در مرورگر خود باز کنید و میزبان را جایگزین میزبان تهیه شده توسط توسعه کنید. با یک کاربر موجود وارد شوید و سپس دکمه BOOKMARK IT را به نوار ابزار نشانک های مرورگر خود بصورت زیر بکشید:



یک وب سایت به دلخواه خود در مرورگر خود باز کرده و روی نشانک خود کلیک کنید. خواهید دید که یک کادر سفید جدید در وب سایت ظاهر می شود و کلیه تصاویر JPEG موجود در ابعاد بالاتر از 100 x 100 پیکسل را نشان می دهد. باید مانند مثال زیر باشد:



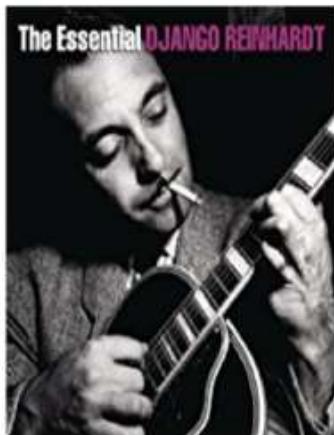
ظرف HTML شامل تصاویری است که می توانند نشانه گذاری شوند. ما می خواهیم کاربر بر روی تصویر دلخواه کلیک کرده و آن را علامت گذاری کند. پرونده استاتیک js / bookmarklet.js را ویرایش کنید و کد زیر را در پایین عملکرد () نشانک اضافه کنید:

```
// when an image is selected open URL with it
jQuery('#bookmarklet .images a').click(function(e){
    selected_image = jQuery(this).children('img').attr('src');
    // hide bookmarklet
    jQuery('#bookmarklet').hide();
    // open new window to submit the image
    window.open(site_url +'images/create/?url='
        + encodeURIComponent(selected_image)
        + '&title='
        + encodeURIComponent(jQuery('title').text()),
        '_blank');
});
```

کد قبلی به شرح زیر عمل می کند:

1. ما یک رویداد کلیک () را به عناصر پیوند تصاویر وصل می کنیم.
2. هنگامی که کاربر روی یک تصویر کلیک می کند ، ما یک متغیر جدید به نام select_image تنظیم می کنیم که حاوی URL تصویر انتخاب شده است.
3. ما نشانک را مخفی می کنیم و برای نشانه گذاری یک تصویر جدید در سایت خود ، یک پنجره مرورگر جدید با URL باز می کنیم. ما عنصر <title> وب سایت و URL تصویری انتخاب شده را به عنوان پارامترهای GET می گذرانیم.
- یک URL جدید را با مرورگر خود باز کنید و دوباره روی نشانک خود کلیک کنید تا قادر انتخاب تصویر نمایش داده شود. اگر بر روی یک تصویر کلیک کنید ، به صفحه ایجاد تصویر هدایت می شوید و عنوان وب سایت و URL آدرس انتخاب شده را به عنوان پارامترهای GET منتقل می کنید:

Bookmark an image



Title:

Django Reinhardt

Description:

تبریک می‌گویم! این اولین نشانک جاوا اسکریپت شماست و کاملاً در پروژه Django شما ادغام شده است.

ایجاد نمای جزئی برای تصاویر

اکنون برای نمایش تصویری که در سایت ما ذخیره شده است، یک نمای جزئیات ساده ایجاد خواهیم کرد. پرونده views.py برنامه تصاویر را باز کنید و کد زیر را به آن اضافه کنید:

```
from django.shortcuts import get_object_or_404
from .models import Image

def image_detail(request, id, slug):
    image = get_object_or_404(Image, id=id, slug=slug)
    return render(request,
                  'images/image/detail.html',
                  {'section': 'images',
                   'image': image})
```

این یک نمایش ساده برای نمایش یک تصویر است. پرونده urls.py برنامه تصاویر را ویرایش کنید و الگوی URL زیر را اضافه کنید:

```
path('detail/<int:id>/<slug:slug>/',
      views.image_detail, name='detail'),
```

پرونده model.py برنامه تصاویر را ویرایش کنید و روش get_absolute_url() را به مدل تصویر اضافه کنید، به شرح زیر:

```
from django.urls import reverse

class Image(models.Model):
    # ...
    def get_absolute_url(self):
        return reverse('images:detail', args=[self.id, self.slug])
```

به پاد داشته باشید که الگوی رایج برای تهیه URL های کانونی برای اشیاء تعريف یک روش `get_absolute_url()` در مدل است.

سرانجام ، یک الگوی درون فهرست / image / images / image.html ایجاد کنید و آنرا `base.html` بنامید. کد زیر را به آن اضافه کنید:

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

{% block title %}{{ image.title }}{% endblock %}

{% block content %}
<h1>{{ image.title }}</h1>

{% with total_likes=image.users_like.count %}
<div class="image-info">
<div>
<span class="count">
{{ total_likes }} like{{ total_likes|pluralize }}
</span>
</div>
{{ image.description|linebreaks }}
</div>
<div class="image-likes">
{% for user in image.users_like.all %}
<div>

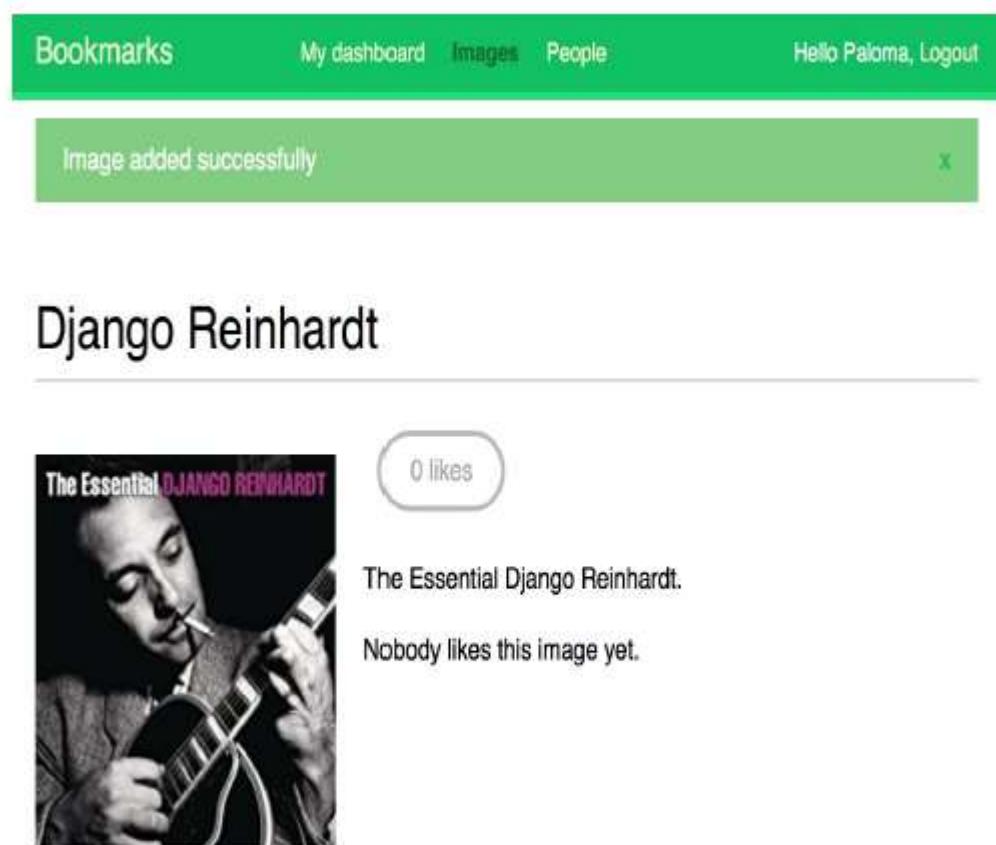
<p>{{ user.first_name }}</p>
</div>
{% empty %}
    Nobody likes this image yet.
{% endfor %}
</div>
{% endwith %}
{% endblock %}
```

این الگوی برای نمایش جزئیات تصویر نشانک است.

ما برای ذخیره کردن نتیجه `QuerySet` از تگ `{% with %}` استفاده می کنیم ، و همه لایک های کاربر را در یک متغیر جدید به نام `total_likes` حساب می کنیم. با این کار ، ما از ارزیابی دوبار `QuerySet` یکسان جلوگیری می کنیم. ما همچنین

توضیحات تصویر را درج می کنیم و آن را بیش از image.users_like.all قرار می دهیم تا تمام کاربرانی که این تصویر را دوست دارند نمایش داده شود.

اکنون با استفاده از نشانک ، یک تصویر جدید را علامت گذاری کنید. بعد از ارسال تصویر به صفحه جزئیات تصویر هدایت می شوید. این صفحه شامل یک پیام موفقیت آمیز است ، به شرح زیر:



ایجاد تصاویر کوچک تصویر با استفاده از تصاویر کوچک

تصویر اصلی را در صفحه جزئیات نمایش می دهیم ، اما ابعاد تصاویر مختلف ممکن است متفاوت باشد. همچنین ، ممکن است پرونده های اصلی برخی از تصاویر بسیار زیاد باشد و بارگذاری آنها ممکن است خیلی طولانی طول بکشد. بهترین راه

برای نمایش تصاویر بهینه شده با روشی یکنواخت، تولید تصاویر کوچک است. برای این منظور از یک برنامه Django به نام sorl-thumbnail استفاده خواهیم کرد.

ترمینال را باز کنید و با استفاده از دستور زیر تصویر زیر را نصب کنید:

```
pip install sorl-thumbnail==12.4.1
```

پرونده py.setting را ویرایش کنید و sorl.thumbnail را به تنظیمات INSTALLED_APPS اضافه کنید،
به شرح زیر:

```
INSTALLED_APPS = [  
    # ...  
    'sorl.thumbnail',  
]
```

سپس دستور زیر را برای همگام سازی برنامه با بانک اطلاعاتی خود اجرا کنید:

```
python manage.py migrate
```

باید خروجی را مشاهده کنید که شامل خط زیر است:

```
Applying thumbnail.0001_initial... OK
```

برنامه sorl-thumbnail روش های مختلفی را برای تعریف تصاویر کوچک تصویر ارائه می دهد. اگر می خواهید تصاویر کوچک را در مدل های خود تعریف کنید، برنامه برای ایجاد ریز عکسها در قالب ها و یک ImageField سفارشی، یک برچسب قالب { % thumbnail image / detail.html } ایجاد می کند. ما از روش برچسب قالب استفاده خواهیم کرد. الگوی / images

```

```

خطوط زیر باید جایگزین حالت قبلی شود:

```
{% load thumbnail %}
{% thumbnail image.image "300" as im %}
<a href="{{ image.image.url }}>
  
</a>
{% endthumbnail %}
```

در اینجا ، یک تصویر کوچک با عرض ثابت 300 پیکسل را تعریف می کنیم. اولین باری که کاربر در این صفحه بارگیری می کند ، یک تصویر کوچک ایجاد می شود.

تصویر کوچک تولید شده در درخواست های زیر ارائه می شود.

서ور توسعه را با دستور `python management.py runserver` شروع کرده و به یک صفحه جزئیات تصویر برای یک تصویر موجود دسترسی پیدا کنید.

تصویر کوچک در سایت ایجاد و نمایش داده می شود.

برنامه `sorl-thumbnail` گزینه های مختلفی برای سفارشی کردن ریز عکسها از جمله الگوریتم های برش و جلوه های مختلفی که قابل استفاده است ارائه می دهد. اگر در ایجاد ریز عکسها مشکلی دارید ، می توانید برای به دست آوردن اطلاعات اشکال زدایی ، `settings.py` را در پرونده `THUMBNAIL_DEBUG = True` صحیح به `True` کنید. می توانید مستندات کامل برنامه `sorl-thumbnail` را در اینجا بخوانید

<https://sorl-thumbnail.readthedocs.io>

افزودن اقدامات jQuery با AJAX

اکنون ، اقدامات AJAX را به برنامه خود اضافه خواهیم کرد. AJAX از JavaScript و XML ناهمزمان است. این اصطلاح شامل گروهی از تکنیک ها برای درخواست های ناهمزمان HTTP است. این شامل ارسال و بازیابی داده ها از سرور به صورت

همزمان و بدون بارگذاری مجدد کل صفحه است. با وجود نام XML لازم نیست. می توانید داده ها را با قالب های دیگر مانند JSON، HTML یا متن ساده ارسال یا بازیابی کنید.

ما پیوندی را به صفحه جزئیات تصویر اضافه می کنیم تا کاربران بتوانند بر روی آن کلیک کنند تا یک تصویر را دوست داشته باشند. ما این کار را با یک تماس AJAX انجام خواهیم داد تا از بارگیری مجدد کل صفحه جلوگیری شود. در ابتدا ، ما نمایشی را برای کاربران ایجاد می کنیم که دوست دارند / برخلاف تصاویر ایجاد کنند. پرونده views.py برنامه تصاویر را ویرایش کنید و کد زیر را به آن اضافه کنید:

```
from django.http import JsonResponse
from django.views.decorators.http import require_POST

@login_required
@require_POST
def image_like(request):
    image_id = request.POST.get('id')
    action = request.POST.get('action')
    if image_id and action:
        try:
            image = Image.objects.get(id=image_id)
            if action == 'like':
                image.users_like.add(request.user)
            else:
                image.users_like.remove(request.user)
            return JsonResponse({'status': 'ok'})
        except:
            pass
    return JsonResponse({'status': 'ko'})
```

برای نمایش خود از دو دکوراتور استفاده خواهیم کرد. تزئین کننده login_required از دسترسی کاربرانی که به سیستم وارد نشده اند جلوگیری می کند. اگر درخواست HTTP از طریق POST انجام نشده باشد ، require_POST یک شیء HttpResponseNotAllowed (کد وضعیت 405) را برمی گرداند. به این ترتیب ، ما فقط درخواست های POST را برای این نمایش مجاز می کنیم. Django همچنین یک دکوراتور requ_GET را فراهم می کند تا فقط به درخواست های GET و یک تزئین need_http_methods پردازد که می توانید لیستی از روش های مجاز را به عنوان آرگومان منتقل کنید.

در این نمای ، ما از دو پارامتر GET استفاده می کنیم:

: شناسه شیء تصویر که کاربر در آن عمل را انجام می دهد image_id

عملی که کاربر می خواهد آن را انجام دهد ، که فرض می کنیم رشته ای با مقداری مانند یا مانند آن نیست

ما از مدیر ارائه شده توسط Django برای قسمت های بسیاری از کاربران مدل تصویر استفاده می کنیم تا اشیاء را از رابطه با استفاده از روش add() یا remove() حذف یا add کنیم. فراخوانی add() یعنی عبور از شیئی که از قبل در مجموعه اشیاء مرتبط وجود دارد ، آن را کپی نمی کند و ، remove() عبور دادن از یک شی که در مجموعه شی مرتبط نیست. روش مفید دیگر مدیر clear() است ، که همه اشیاء را از مجموعه شی مرتبط بربط می دهد. سرانجام ، ما از کلاس JsonResponse تهیه شده توسط Django استفاده می کنیم ، که یک پاسخ HTTP را با نوع برنامه / JSON محتوای برمی گرداند و شیء داده شده را به یک خروجی JSON تبدیل می کند.

پرونده urls.py برنامه تصاویر را ویرایش کنید و الگوی URL زیر را به آن اضافه کنید:

```
path('like/', views.image_like, name='like'),
```

بارگیری jQuery

ما باید عملکرد AJAX را به الگوی جزئیات تصویر خود اضافه کنیم. برای استفاده از jQuery در قالب های ما ، ابتدا آن را در قالب base.html پژوهه خود می گنجانیم. الگوی base.html برنامه حساب را ویرایش کنید و قبل از بسته شدن کد زیر را وارد کنید. <body/> برجسب HTML:

```
<script
src="https://ajax.googleapis.com/ajax/libs/jquery/3.2.1/jquery.min.js">
</script>
<script>
$(document).ready(function(){
    {% block domready %}
    {% endblock %}
});
</script>
```

ما چارچوب /<https://jquery.com> بارگیری می کنیم. همچنین می توانید jQuery را از CDN Google بارگیری کرده و در عوض آن را به فهرست استاتیک برنامه خود اضافه کنید. ما برای اضافه کردن کد جاوا اسکریپت ، یک بروچسب <script> اضافه می کنیم. \$(document).ready() یک تابعjQuery است که یک دستگیرنده را اجرا می کند که با انجام سلسله مراتب DOM به طور کامل ساخته شده است. از Model Object Model DOM تهیه می شود. هنگام مرور صفحه وب توسط مرورگر ایجاد می شود و به عنوان درخت اشیاء ساخته می شود.

با وارد کردن کد ما در داخل این عملکرد ، ما اطمینان حاصل خواهیم کرد که تمام عناصر HTML که می خواهیم با آنها ارتباط برقرار کنیم در DOM بارگذاری شده است. کد ما فقط پس از آماده شدن DOM اجرا می شود.

در داخل تابع کنترل کننده سند آماده ، یک بلوک الگوی جنگو به نام داخلی را شامل می شود ، که در آن قالب هایی که قالب پایه را گسترش می دهند می توانند JavaScript خاص را شامل شوند.

با بروچسب های جاوا اسکریپت و بروچسب الگوی جنگو سردرگم نشود. زیان قالب جنگو در سمت سرور ارائه می شود که سند HTML نهایی را به دست می آورد و جاوا اسکریپت را در سمت مشتری اجرا می کند. در برخی موارد ، تولید کد JavaScript به صورت پویا با استفاده از جنگو مفید است.

در مثالهای این فصل ، کد JavaScript را در قالب های جنگو درج می کنیم. روش برتر برای درج کد جاوا اسکریپت بارگیری پرونده های .js است که به عنوان فایلهای استاتیک بخصوص هنگایی که اسکریپت های بزرگ هستند ارائه می شوند.

درخواست متقابل درخواست جعلی در درخواست های AJAX

با فعال بودن حفاظت CSRF ، Django در تمام درخواستهای POST یک نشانه CSRF را بررسی می کند. هنگام ارسال فرم ، می توانید از قالب % csrf_token استفاده کنید تا نشانه را به همراه فرم ارسال کنید. با این حال ، برای درخواستهای AJAX کمی ناخوشایند است که بتواند با هر درخواست POST ، نشانه CSRF را به عنوان داده POST در بیاورد. بنابراین ، Django به شما امکان می دهد تا یک هدر X-CSRFToken در درخواست های AJAX خود با ارزش نشانه CSRF تنظیم کنید. این به شما امکان می دهد jQuery یا هر کتابخانه جاوا اسکریپت دیگری تنظیم کنید تا به طور خودکار هدر-X-CSRFToken را در هر درخواست تنظیم کنید.

برای درج کد درخواست ها ، باید مراحل زیر را انجام دهید:

1- نشانه CSRF را از کوکی csrfmiddlewaretoken بازیابی کنید ، که در صورت فعال بودن محافظت در CSRF است

2. علامت گذاری در درخواست AJAX را با استفاده از هدر X-CSRFToken ارسال کنید

می توانید اطلاعات بیشتر در مورد حفاظت CSRF و AJAX را در

می توانید اطلاعات بیشتر در مورد حفاظت CSRF و AJAX را در <https://docs.djangoproject.com/en/2.0/ref/csrf/#ajax> پیدا کنید.

آخرین کدی را که در الگوی base.html خود درج کرده اید ویرایش کنید و به صورت زیر ظاهر کنید:

```
<script  
src="https://ajax.googleapis.com/ajax/libs/jquery/3.2.1/jquery.min.js">  
</script>  
<script src="https://cdn.jsdelivr.net/npm/js-cookie@2/src/js.cookie.min.js">  
</script>  
<script>  
    var csrftoken = Cookies.get('csrftoken');  
    function csrfSafeMethod(method) {  
        // these HTTP methods do not require CSRF protection  
        return (/^(GET|HEAD|OPTIONS|TRACE)$/.test(method));  
    }  
    $.ajaxSetup({  
        beforeSend: function(xhr, settings) {  
            if (!csrfSafeMethod(settings.type) && !this.crossDomain) {  
                xhr.setRequestHeader("X-CSRFToken", csrftoken);  
            }  
        }  
    });  
    $(document).ready(function(){  
        {% block domready %}  
        {% endblock %}  
    });  
</script>
```

کد قبلی به شرح زیر است:

1. ما افزونه JS Cookie را از CDN عمومی بارگیری می کنیم تا بتوانیم به راحتی با کوکی ها ارتباط برقرار کنیم. JS Cookie جواوا اسکریپت سبک برای کار با کوکی ها است. می توانید اطلاعات بیشتری در مورد آن در <https://github.com/js-cookie/js-cookie> کسب کنید.
2. مقدار کوکی Cookies.get را با csrftoken می خوانیم () .
3. ما عملکرد csrfSafeMethod () را برای بررسی این نیازی به HTTP روشن بودن روشن تعییف می کنیم. روش های این نیازی به محافظت CSRF ندارند: اینها GET، HEAD، OPTIONS و TRACE هستند.

4. ما درخواست های جی کوئری AJAX را با استفاده از `\$().ajaxSetup()` تنظیم کرده ایم. قبیل از انجام هر درخواست AJAX ، بررسی می کنیم که آیا روش درخواست بی خطر است و درخواست فعلی crossdomain نیست.

اگر درخواست نامن است ، عنوانی X-CSRFToken را با مقدار به دست آمده از کوکی تنظیم می کنیم. این تنظیم برای کلیه درخواستهای AJAX انجام شده با `jQuery` اعمال خواهد شد.

نشان CSRF در کلیه درخواستهای AJAX که از روش‌های HTTP غیر ایمن مانند POST یا PUT استفاده می کنند ، درج خواهد شد.

انجام درخواستهای AJAX با jQuery

الگوی تصاویر / `image / detail.html` کاربرد تصاویر را ویرایش کنید و خط زیر را در نظر بگیرید:

```
{% with total_likes=image.users_like.count %}
```

مورد قبلی را با موارد زیر جایگزین کنید:

```
{% with total_likes=image.users_like.count users_like=image.users_like.all %}
```

سپس عنصر `<div>` را با کلاس `image-info` تغییر دهید ، به شرح زیر:

```

<div class="image-info">
  <div>
    <span class="count">
      <span class="total">{{ total_likes }}</span>
      like{{ total_likes|pluralize }}
    </span>
    <a href="#" data-id="{{ image.id }}" data-action="{% if
request.user in users_like %}unlike{% endif %}like"
class="like button">
      {% if request.user not in users_like %}
        Like
      {% else %}
        Unlike
      {% endif %}
    </a>
  </div>
  {{ image.description|linebreaks }}
</div>

```

ابتدا ، به منظور ذخیره نتایج جستجوی all image.users_like tag به `{% with %}` اضافه کردیم و از اجرای آن دو بار خودداری کرد. ما تعداد کل کاربرانی که این تصویر را دوست دارند نمایش می دهیم و برای تصویر like/unlike را شامل می شویم: بررسی می کنیم که کاربر در مجموعه شی مربوط به کاربرانی باشد که دوست دارد یا بر خلاف آن ، براساس روابط فعلی کاربر نمایش دهد. و این تصویر ما عناصر زیر را به عنصر `<a>` اضافه می کنیم:

data-id: شناسه تصویر نمایش داده شده عمل: عملی که با کلیک کاربر روی پیوند ، اجرا شود. این می تواند مانند یا بر خلاف باشد ، ما مقدار درخواست هر دو ویژگی را در درخواست image_like AJAX به نمای ارسال خواهیم کرد. هنگامی که کاربر روی پیوند مانند / برخلاف کلیک کرد ، ما باید اقدامات زیر را از طرف مشتری انجام دهیم:

1. با عبور از شناسه AJAX و شناسه تصویر و پارامترهای عمل به آن تماس بگیرید.
2. اگر درخواست AJAX موفقیت آمیز بود ، ویژگی عمل داده را از عنصر `<a>` با عمل (like/unlike) به روز کنید ، و متن نمایش آن را مطابق با آن تغییر دهید.
3. تعداد لایک های نمایش داده شده را به روز کنید.

بلوک موجود در انتهای الگوی images / image / detail.html زیر اضافه کنید:

```
{% block domready %}
$('a.like').click(function(e){
  e.preventDefault();
  $.post('{% url "images:like" %}',
  {
    id: $(this).data('id'),
    action: $(this).data('action')
  },
  function(data){

    if (data['status'] == 'ok')
    {
      var previous_action = $('a.like').data('action');

      // toggle data-action
      $('a.like').data('action', previous_action == 'like' ?
        'unlike' : 'like');
      // toggle link text
      $('a.like').text(previous_action == 'like' ? 'Unlike' :
        'Like');

      // update total likes
      var previous_likes = parseInt($('span.count .total').text());
      $('span.count .total').text(previous_action == 'like' ?
        previous_likes + 1 : previous_likes - 1);
    }
  }
));
});
{% endblock %}
```

کد قبلی به شرح زیر عمل می کند:

1. ما از انتخاب کننده \$('a.like') برای یافتن همه عناصر <a> سند HTML با کلاس مشابه استفاده می کنیم.
 2. ما یک عملکرد کنترل کننده را برای رویداد کلیک تعريف می کنیم. این عملکرد هر بار که کاربر روی like/unlike کلیک کند، اجرا می شود.
 3. در داخل تابع e.preventDefault ، handler استفاده می کنیم. این امر مانع از پیوند ما در هر نقطه می شود.
 - 4- ما از \$ post() برای انجام درخواست POST غیر همزمان به سرور استفاده می کنیم. jQuery همچنین یک متده get() را برای انجام درخواست GET و یک روش پایین \$.ajax سطح پایین ارائه می دهد.
 - 5- برای ایجاد URL برای درخواست AJAX از تگ قالب { URL % } استفاده می کنیم.
 6. ما فرهنگ نامه پارامترهای POST را برای ارسال درخواست ایجاد می کنیم. این پارامترهای شناسایی و عملکرد هستند که انتظار می رود
- نمای جنگو. ما این مقادیر را از خصوصیات <a> و عملکرد داده داده ها بازیابی می کنیم.
7. ما یک تابع پاسخ به کار را تعريف می کنیم که وقتی انجام می شود پاسخ HTTP دریافت می شود که یک ویژگی داده ای را شامل می شود که حاوی محتوای پاسخ است.
 - 8- ما به ویژگی وضعیت داده های دریافت شده دسترسی می یابیم و بررسی می کنیم که آیا این برابر است یا خیر. اگر داده های برگشتی همانطور که انتظار می رود ما ویژگی عمل داده پیوند و متن آن را تغییر می دهیم. این به کاربر اجازه می دهد عملکرد خود را خنثیسازی کند.
 - 9- بسته به عملی که انجام می دهیم تعداد کل لایک ها را یک یا یک کاهش می دهیم یا کاهش می دهیم.
- برای تصویری که آپلود کرده اید ، صفحه جزئیات تصویر را در مرورگر خود باز کنید. باید بتوانید تعداد لایک های اولیه زیر و دکمه LIKE را به شرح زیر مشاهده کنید: نمای Django ما این مقادیر را از عناصر <id> عنصر داده و عملکرد داده عنصر بازیابی می کنیم.
7. ما یک عملکرد برگشتی را تعريف می کنیم که هنگام دریافت پاسخ HTTP انجام می شود. طول می کشد یک ویژگی داده ای که شامل محتوای پاسخ است.

8- ما به ویژگی وضعیت داده های دریافت شده دسترسی داریم و بررسی می کنیم که آیا این برابر است یا خیر. اگر داده های برگشتی همانطور که انتظار می رود ، ویژگی عملکرد داده پیوند و متن آن را تغییر می دهیم. این به کاربر اجازه می دهد عملکرد خود را خنثیسازی کند.

9- بسته به عملی که انجام می دهیم تعداد کل لایک ها را یک یا یک کاهش می دهیم یا کاهش می دهیم.
برای تصویری که آپلود کرده اید ، صفحه جزئیات تصویر را در مرورگر خود باز کنید. شما باید بتوانید تعداد لایک های اولیه و دکمه LIKE را به شرح زیر مشاهده کنید:



هنگامی که بر روی دکمه UNLIKE کلیک می کنید ، عمل انجام می شود ، متن دکمه به LIKE تغییر می یابد و تعداد کل بر این اساس تغییر می کند.

هنگام برنامه نویسی JavaScript ، به خصوص هنگام انجام درخواستهای AJAX ، توصیه می شود از ابزاری برای اشکال زدایی در درخواست های JavaScript و HTTP استفاده کنید. بیشتر مرورگرهای مدرن شامل ابزارهای برنامه نویس برای اشکالزدایی در JavaScript هستند. معمولاً می توانید در هر جای وب سایت کلیک راست کرده و بر روی عنصر Inspect کلیک کنید تا به ابزارهای توسعه دهنده وب دسترسی پیدا کنید.

ایجاد دکوراتورهای سفارشی برای نماهای شما

ما نمایهای AJAX را محدود خواهیم کرد تا فقط درخواستهای ایجاد شده از طریق AJAX انجام شود. هدف درخواست XMLHttpRequest() را فراهم می کند که بررسی می کند آیا درخواست با Django یعنی درخواست AJAX است. این مقدار در عنوان HTTP_X_REQUESTED_WITH تنظیم شده است ، که در درخواست های AJAX توسط بیشتر کتابخانه های جاوا اسکریپت گنجانده شده است.

ما یک نماینده را برای بررسی عنوان HTTP_X_REQUESTED_WITH در نماهای خود ایجاد خواهیم کرد. دکوراتور تابعی است که عملکرد دیگری را به خود اختصاص می دهد و بدون تغییر صریح آن رفتار دومی را گسترش می دهد. اگر مفهوم دکوراتور برای شما خارجی است ، ممکن است قبل از ادامه خواندن ، به <https://www.python.org/dev/peps/pep-0318/> نگاهی بیندازید.

از آنجایی که دکوراتور ما عمومی است و می تواند برای هر نظر اعمال شود ، ما یک بسته مشترک پایتون را در پروژه خود ایجاد خواهیم کرد. دایرکتوری و پرونده های زیر را در فهرست پروژه نشانک ها ایجاد کنید:

```
common/  
    __init__.py  
decorators.py
```

پرونده decorators.py را ویرایش کنید و کد زیر را به آن اضافه کنید:

```
from django.http import HttpResponseRedirect  
  
def ajax_required(f):  
    def wrap(request, *args, **kwargs):  
        if not request.is_ajax():  
            return HttpResponseRedirect()  
  
        return f(request, *args, **kwargs)  
    wrap.__doc__=f.__doc__  
    wrap.__name__=f.__name__  
    return wrap
```

کد قبلی دکوراتور سفارشی `ajax_required` ما است. این یک تابع بسته بندی را تعریف می کند که اگر درخواست AJAX نباشد ، یک شی `HttpResponseBadRequest` بازگرداند. در غیر این صورت ، عملکرد تزئین شده را برگرداند.

اکنون می توانید پرونده `views.py` برنامه تصاویر را ویرایش کرده و این دکوراتور را به نمای AJAX خود به شکل زیر اضافه کنید:

```
from common.decorators import ajax_required

@ajax_required
@login_required
@require_POST
def image_like(request):
    # ...
```

اگر سعی دارید به طور مستقیم با مرورگر خود به `http://127.0.0.1:8000/images/like` / دسترسی پیدا کنید ، یک پاسخ HTTP 400 دریافت خواهد کرد.

افزودن صفحه بندی AJAX به نمای لیست شما

ما نیاز به لیست تمام تصاویر نشانک شده در وب سایت خود داریم. ما برای ایجاد قابلیت های پیمایش نامتناهی از برنامه ریزی AJAX استفاده خواهیم کرد. پیمایش نامحدود با بارگیری نتایج بعدی به طور خودکار هنگام حرکت کاربر به پایین صفحه ، حاصل می شود.

ما نمای لیست لیست را اجرا خواهیم کرد که هم به درخواست های استاندارد مرورگر و هم با درخواست های AJAX از جمله صفحه بندی رسیدگی می کند. وقتی کاربر در ابتدا صفحه لیست تصویر را بارگیری می کند ، ما صفحه اول تصاویر را نمایش می دهیم. هنگامی که آنها به پایین صفحه می روند ، ما صفحه زیر موارد را از طریق AJAX بارگذاری می کنیم و آن را در پایین صفحه اصلی ضمیمه می کنیم.

همین نمای صفحه با صفحات استاندارد و AJAX روبرو خواهد شد. پرونده views.py برنامه تصاویر را ویرایش کنید و کد زیر را به آن اضافه کنید:

```
from django.http import HttpResponseRedirect
from django.core.paginator import Paginator, EmptyPage, \
    PageNotAnInteger

@login_required
def image_list(request):
    images = Image.objects.all()
    paginator = Paginator(images, 8)
    page = request.GET.get('page')
    try:
        images = paginator.page(page)
    except PageNotAnInteger:
        # If page is not an integer deliver the first page
        images = paginator.page(1)
    except EmptyPage:
        if request.is_ajax():
            # If the request is AJAX and the page is out of range
            # return an empty page
            return HttpResponseRedirect('')
        # If page is out of range deliver last page of results
        images = paginator.page(paginator.num_pages)
    if request.is_ajax():
        return render(request,
                     'images/image/list_ajax.html',
                     {'section': 'images', 'images': images})
    return render(request,
                 'images/image/list.html',
                 {'section': 'images', 'images': images})
```

در این نمای ، ما QuerySet ایجاد می کنیم تا همه تصاویر از پایگاه داده بازگردد. سپس ، برای دستیابی به نتایج ، یک شیء Paginator ایجاد می کنیم و هشت تصویر در هر صفحه را بازیابی می کنیم. اگر صفحه درخواستی خارج از محدوده باشد ، یک استثناء صفحه خالی دریافت می کنیم. اگر این مورد باشد و درخواست از طریق AJAX انجام شود ، ما یک XMLHttpRequest خالی را برگردانده می شود که به ما کمک می کند تا صفحه بندی AJAX را در سمت مشتری متوقف کنیم. ما نتایج را به دو قالب مختلف ارائه می دهیم:

برای درخواست های AJAX ، از الگوی list_ajax.html ارائه می دهیم. این الگو فقط شامل تصاویر صفحه درخواستی خواهد بود. برای درخواست های استاندارد ، الگوی list.html را ارائه می دهیم. این قالب base.html را برای نمایش کل صفحه گسترش می دهد و شامل قالب list_ajax.html می شود تا لیست تصاویر را درج کند.

پرونده urls.py برنامه تصاویر را ویرایش کنید و الگوی URL زیر را به آن اضافه کنید:

```
path('', views.image_list, name='list'),
```

سرانجام ، ما نیاز به ایجاد الگوهای ذکر شده در اینجا داریم. در داخل فهرست /images/image کنید و آن را list_ajax.html بنامید. کد زیر را به آن اضافه کنید:

```
{% load thumbnail %}

{% for image in images %}
<div class="image">
    <a href="{{ image.get_absolute_url }}>
        {% thumbnail image.image "300x300" crop="100%" as im %}
            <a href="{{ image.get_absolute_url }}>
                
        <a href="{{ image.get_absolute_url }}" class="title">
            {{ image.title }}
        </a>
    </div>
</div>
{% endfor %}
```

الگوی قبلی لیست تصاویر را نشان می دهد. ما از آن برای بازگشت نتایج برای درخواستهای AJAX استفاده خواهیم کرد.
الگوی دیگری را در همان فهرست ایجاد کنید و نام آن را لیست کنید. کد زیر را به آن اضافه کنید:

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

{% block title %}Images bookmarked{% endblock %}

{% block content %}
    <h1>Images bookmarked</h1>
    <div id="image-list">
        {% include "images/image/list_ajax.html" %}
    </div>
{% endblock %}
```

الگوی لیست الگوی base.html را گسترش می دهد. برای جلوگیری از تکرار کد ، از الگوی list_ajax.html برای نمایش تصاویر درج کردیم. الگوی list.html کد جاوا اسکریپت را برای بارگیری صفحات اضافی هنگام پیمایش به پایین صفحه ، نگه می دارد.

کد زیر را به الگوی list.html اضافه کنید:

```
{% block domready %}
```

```
var page = 1;
var empty_page = false;
var block_request = false;

$(window).scroll(function() {
    var margin = $(document).height() - $(window).height() - 200;
    if ($("window").scrollTop() > margin && empty_page == false &&
        block_request == false) {
        block_request = true;
        page += 1;
        $.get('?page=' + page, function(data) {
            if(data == '') {
                empty_page = true;
            }
            else {
                block_request = false;
                $('#image-list').append(data);
            }
        });
    }
});
{% endblock %}
```

کد قبلی قابلیت پیمایش نامتناهی را ارائه می دهد. ما کد JavaScript را در بلوک dom-ready که در الگوی base.html کد قابلیت پیمایش نامتناهی را ارائه می دهد. تعریف کرده ایم درج می کنیم. کد به شرح زیر است:

1. متغیرهای زیر را تعریف می کنیم:

1. صفحه: شماره صفحه فعلی را ذخیره می کند.

2. empty_page: به ما این امکان را می دهد که بدانیم کاربر در صفحه آخر قرار دارد یا یک صفحه خالی را بازیابی می کند. به محض اینکه یک صفحه خالی به دست آوردهیم ، ما ارسال درخواست های اضافی AJAX را متوقف خواهیم کرد زیرا فرض خواهیم کرد که نتیجه دیگری حاصل نشده است.

3. block_quest: در حالی که یک درخواست AJAX در حال انجام است ، مانع ارسال درخواست های اضافی نمی شود.

2. ما برای گرفتن رویداد پیمایش از (scroll) winsows استفاده می کنیم و همچنین یک عملکرد کنترل کننده را برای آن تعریف می کنیم.

3. متغیر حاشیه را محاسبه می کنیم تا تفاوت بین کل طول سند و ارتفاع پنجره را بدست آوریم ، زیرا این مقدار محتوای باقی مانده برای پیمایش کاربر است. ما مقدار 200 را از نتیجه کم می کنیم تا وقتی کاربر نزدیک به 200 پیکسل به انتهای صفحه بآمد ، صفحه بعدی را بازگذاری کنیم.

4- ما فقط درخواست AJAX را ارسال می کنیم در صورتی که هیچ درخواست AJAX دیگری انجام نشده باشد empty_page is also false (block_request has to be false)

5- ما برای جلوگیری از وضعیتی که رویداد پیمایش درخواستهای اضافی AJAX را ایجاد کند ، شمارنده صفحه را یک به یک افزایش داده و برای بازیابی صفحه بعدی ، تعداد block _ درخواست را صحیح تنظیم می کنیم.

6. ما درخواست GET AJAX را با استفاده از ()\$.get انجام می دهیم و پاسخ HTML را در متغیری به نام داده دریافت می کنیم. در زیر دو سناریو وجود دارد:

1. پاسخ هیچ محتوایی ندارد: ما به انتهای نتایج رسیدیم و هیچ صفحه دیگری برای بازگیری وجود ندارد.

برای جلوگیری از درخواست های اضافی AJAX ، ما empty_page را درست قرار دادیم.

2. پاسخ حاوی داده ها است: ما داده ها را با شناسه لیست تصویر به عنصر HTML اضافه می کنیم. وقتی کاربر به انتهای صفحه نزدیک شد ، محتوای صفحه به طور عمودی نتایج ضمیمه را گسترش می دهد.

/ را در مرورگر خود باز کنید. لیست تصاویری که تاکنون بوک مارک گذاری کرده اید را مشاهده خواهید کرد. باید شبیه به این باشد:

Bookmarks My dashboard Images People Hello Antonio, Logout

Images bookmarked

			
Louis Armstrong	Chick Corea	Al Jarreau	Al Jarreau
			

برای بارگیری صفحات اضافی به انتهای صفحه بروید. اطمینان حاصل کنید که بیش از هشت تصویر را با استفاده از نشانک علامت گذاری کرده اید زیرا این تعداد تصویری است که در هر صفحه نمایش می دهیم. به یاد داشته باشید که می توانید از Firebug یا ابزار مشابهی برای ردیابی درخواستهای AJAX استفاده کرده و کد جاوا اسکریپت خود را اشکال زدایی کنید.

در آخر ، الگوی base.html برنامه حساب کاربری را ویرایش کنید و آدرس URL تصاویر مورد اصلی منورا به شرح زیر اضافه کنید:

```
<li {% if section == "images" %}class="selected"{% endif %}>
  <a href="{% url "images:list" %}">Images</a>
</li>
```

اکنون می توانید از فهرست اصلی به لیست تصویر دسترسی پیدا کنید.

خلاصه

در این فصل ما یک نشانک جاوا اسکریپت ایجاد کرده ایم تا تصاویر سایر وب سایت ها را در سایت خود به اشتراک بگذاریم. شما نماهای AJAX را با jQuery پیاده سازی کرده اید و صفحه بندهی AJAX را اضافه کرده اید.

در فصل بعدی نحوه ساختن یک سیستم پیروی و یک جریان فعالیت به شما آموزش خواهیم داد. شما با روابط عمومی ، سیگنالها و ناهنجاری سازی کار خواهید کرد. شما همچنین یاد می گیرید که چگونه از Django با Redis استفاده کنید.

فصل ششم

پیگیری عملکردهای کاربر

در فصل قبل ، شما نمایه های AJAX را با استفاده از jQuery به پروژه خود پیاده سازی کرده اید و یک نشانک جاوا

اسکریپت برای به اشتراک گذاشتن محتوا از وب سایت های دیگر روی سیستم عامل خود ایجاد کرده اید.

در این فصل نحوه ساختن سیستم پیروی و ایجاد یک جریان فعالیت کاربر را خواهید آموخت. خواهید فهمید که چگونه سیگنال های Django کار می کنند و ذخیره O / I سریع Redis را در پروژه خود ادغام می کنند تا نماهای مورد را ذخیره کنند.

این فصل نکات زیر را در بر می گیرد:

ایجاد روابط بسیار زیاد با یک مدل واسطه

ساختن یک سیستم پیروی

ایجاد یک برنامه جریان فعالیت

افزودن روابط عمومی به مدلها

بهینه سازی QuerySets برای اشیاء مرتبط

استفاده از سیگнал ها برای شمارش آزار دهنده تعداد

ذخیره نماهای مورد در Redis

ساختن یک سیستم دنبال کننده

ما یک سیستم پیرو را در پروژه خود ایجاد خواهیم کرد. کاربران ما قادر به پیگیری یکدیگر و پیگیری آنچه سایر کاربران در این پلتفرم به اشتراک می گذارند هستند. رابطه بین کاربران یک رابطه بسیار زیاد است. یک کاربر می تواند چندین کاربر را دنبال کند و آنها نیز به نوبه خود می توانند توسط چندین کاربر دنبال شوند.

ایجاد روابط بسیار زیاد با یک مدل واسطه

در فصل های قبلی ، شما با اضافه کردن ManyToManyField به یک از مدل های مرتبط ، روابط بسیاری به بسیاری ایجاد کردید و به Django اجازه دادید که جدول پایگاه داده برای این رابطه ایجاد کند. این برای اکثر موارد مناسب است ، اما بعضی اوقات ممکن است شما نیاز به ایجاد یک مدل واسطه برای رابطه داشته باشید. ایجاد یک مدل واسطه زمانی ضروری است که می خواهید اطلاعات اضافی را برای رابطه ذخیره کنید ، به عنوان مثال تاریخ ایجاد رابطه یا زمینه ای که توصیف ماهیت رابطه است.

ما یک مدل واسطه برای ایجاد روابط بین کاربران ایجاد خواهیم کرد. دو دلیل وجود دارد که می خواهیم از یک مدل واسطه استفاده کنیم:

ما از الگوی کاربر ارائه شده توسط Django استفاده می‌کنیم، و می‌خواهیم از تغییر دادن آن جلوگیری کنیم می‌خواهیم زمان ایجاد رابطه را ذخیره کنیم

پرونده model.py برنامه حساب خود را ویرایش کنید و کد زیر را به آن اضافه کنید:

```

        on_delete=models.CASCADE)
created = models.DateTimeField(auto_now_add=True,
                             db_index=True)

class Meta:
    ordering = ('-created',)

def __str__(self):
    return '{} follows {}'.format(self.user_from,
                                  self.user_to)

```

کد قبل ، مدل مخاطبی را که برای روابط کاربر استفاده خواهیم کرد ، نشان می دهد. این قسمت ها شامل موارد زیر است:

برای کاربری که ایجاد رابطه user_to: ForeignKey برای کاربر مورد نظر خود ایجاد می کند:
یک فیلد DateTimeField با auto_now_add = درست برای ذخیره کردن زمان ایجاد رابطه

نمایه بانک اطلاعاتی به طور خودکار در قسمت های ForeignKey ایجاد می شود. ما از db_index = True برای ایجاد یک فهرست بانک اطلاعاتی برای قسمت ایجاد شده استفاده می کنیم.

این عملکرد پرس و جو را هنگام سفارش QuerySets توسط این قسمت بهبود می بخشد.

با استفاده از ORM ، می توانیم برای کاربر — user1 - دنبال کردن یک کاربر دیگر ، user2 ، مانند این رابطه برقرار کنیم:

```

user1 = User.objects.get(id=1)
user2 = User.objects.get(id=2)
Contact.objects.create(user_from=user1, user_to=user2)

```

مدیران مرتبط QuerySet و rel_to_set rel_from_set یک Contact را برای مدل برخی گردانند. برای دسترسی به قسمت انتهایی رابطه از مدل کاربر ، مطلوب است که کاربر شامل ManyToManyField باشد ، به شرح زیر:

```
following = models.ManyToManyField('self',
                                  through=Contact,
                                  related_name='followers',
                                  symmetrical=False)
```

در مثال قبل ، ما به Django می گوییم که از الگوی واسطه سفارشی خود برای ایجاد رابطه با اضافه کردن استفاده کردیم. این یک رابطه بسیار زیاد از مدل کاربر با خود است: ما در زمینه ManyToManyField برای ایجاد رابطه با همان مدل به "خود" مراجعه می کنیم.

اگر مدل User بخشی از برنامه ما بود ، می توانستیم قسمت قبلی را به مدل اضافه کنیم. با این حال ، مانعی توانیم کلاس کاربر را مستقیماً تغییر دهیم زیرا متعلق به برنامه django.contrib.auth است. ما با اضافه کردن این قسمت به صورت پویا به مدل کاربر ، رویکرد کمی متفاوت خواهیم گرفت. پرونده model.py برنامه حساب را ویرایش کنید و خطوط زیر را اضافه کنید:

```
from django.contrib.auth.models import User

# Add following field to User dynamically
User.add_to_class('following',
                  models.ManyToManyField('self',
                                        through=Contact,
                                        related_name='followers',
                                        symmetrical=False))
```

در کد قبلی ، از روش add_to_class () از مدل های جنگو برای میکاپ مدل کاربر استفاده می کنیم. توجه داشته باشید که استفاده از add_to_class () راه پیشنهادی برای اضافه کردن فیلدها به مدل ها نیست. با این وجود ما به دلیل دلایل زیر از استفاده از آن در این مورد استفاده می کنیم:

ما نحوه بازیابی اشیاء مرتبط با استفاده از

Django ORM با user.following.all و user.followers.all. ما از مدل Contact واسطه استفاده می کنیم و از پرس و جوهای پیچیده ای که می تواند پیوندهای اضافی در پایگاه داده باشد ، جلوگیری می کند ، همانطور که اتفاق می افتد ، در صورت تعریف رابطه در مدل Profile custom ما.

جدول این رابطه چند به چند با استفاده از مدل های مخاطب ایجاد می شود که ManyToManyField به صورت پویا اضافه می شود ، حاکی از تغییرات دیتابیس برای مدل کاربر Django نیست.

ما از ایجاد یک مدل کاربر سفارشی خودداری می کنیم ، و تمام مزایای استفاده داخلی کاربر Django را حفظ می کنیم.

به خاطر داشته باشید که در بیشتر موارد ، بهتر است به جای اینکه میکاپ را در مدل کاربر قرار دهید ، زمینه هایی به مدل Profile که قبلًاً ایجاد کرده ایم اضافه کنیم. همچنین به شما امکان می دهد از مدل های کاربر سفارشی استفاده کنید. اگر می خواهید از مدل کاربر سفارشی خود استفاده کنید ، به اسناد در <https://docs.djangoproject.com/fa/2.0/topics/auth/customizing> نگاهی بیندازید

می توانید توجه داشته باشید که این رابطه شامل ManyToManyField=symmetrical=False است. وقتی یک symmetrical=False مدل تعریف می کنید ، جنگو رابطه را مجبور می کند تا متقارن شود. در این حالت ، ما در حال تنظیم symmetrical=False برای تعریف یک رابطه غیر متقارن هستیم. این اگر شما را دنبال کنم ، به این معنی نیست که شما به طور خودکار از من پیروی کنید.

دستورالعمل زیر را برای تولید مهاجرت اولیه برای برنامه account اجرا کنید:

```
python manage.py makemigrations account
```

شما خروجی زیر را بدست می آورید:

```
Migrations for 'account':
  account/migrations/0002_contact.py
    - Create model Contact
```

اکنون ، دستور زیر را برای همگام سازی برنامه با پایگاه داده اجرا کنید:

```
python manage.py migrate account
```

باید خروجی را مشاهده کنید که شامل خط زیر است:

```
Applying account.0002_contact... OK
```

اکنون با بانک اطلاعاتی همگام سازی شده است و ما قادر به ایجاد روابط بین کاربران هستیم. با این حال ، سایت ما هنوز راهی برای مرور کاربران یا دیدن یک پروفایل کاربری خاص ارائه نمی دهد. باید لیست و نمایش جزئیات را برای مدل کاربر ایجاد کنیم.

ایجاد لیست و نمایش جزئیات برای پروفایل کاربر

پرونده views.py برنامه حساب را باز کنید و کد زیر را به آن اضافه کنید:

```
from django.shortcuts import get_object_or_404
from django.contrib.auth.models import User

@login_required
def user_list(request):
    users = User.objects.filter(is_active=True)
    return render(request,
                  'account/user/list.html',
                  {'section': 'people',
                   'users': users})

@login_required
def user_detail(request, username):
    user = get_object_or_404(User,
                            username=username,
                            is_active=True)
    return render(request,
                  'account/user/detail.html',
                  {'section': 'people',
                   'user': user})
```

اینها لیست ساده و نمایش جزئیات برای اشیاء کاربر است. نمای user_list همه کاربران فعال را بدست می آورد. مدل کاربر Django حاوی یک پرچم is_active است تا مشخص شود آیا حساب کاربری فعال است یا خیر. ما کوئی را با استفاده از is_braei بازگشت فقط کاربران فعال فیلتر می کنیم. این نمای کلیه نتایج را بر می گرداند ، اما می توانید با اضافه کردن صفحه بندی به همان روشی که برای نمای image_list انجام دادیم ، آن را بهبود بخشد.

نمای user_detail از میانبر get_object_or_404 () برای بازیابی کاربر فعال با نام کاربری خاص استفاده می کند. در صورت عدم یافتن کاربر فعال با نام کاربری مشخص شده ، این نمایش پاسخ HTTP 404 را بر می گرداند.

پرونده urls.py برنامه حساب را ویرایش کرده و الگوی URL برای هر نمای را به شرح زیر اضافه کنید:

```
urlpatterns = [
    # ...
    path('users/', views.user_list, name='user_list'),
    path('users/<username>', views.user_detail, name='user_detail')
]
```

ما از الگوی URL user_detail برای تولید URL کانونی برای کاربران استفاده خواهیم کرد. شما قبل‌آیک روش get_absolute_url () را در یک مدل تعریف کرده اید تا نشانی اینترنتی کانونی برای هر شی را برگردانید. راه دیگر برای مشخص کردن URL برای یک مدل ، اضافه کردن تنظیم ABSOLUTE_URL_OVERRIDES به پروژه شما است.

پرونده settings.py پروژه خود را ویرایش کنید و کد زیر را به آن اضافه کنید:

```
from django.urls import reverse_lazy

ABSOLUTE_URL_OVERRIDES = {
    'auth.user': lambda u: reverse_lazy('user_detail',
                                         args=[u.username])
}
```

جنگو یک روش ABSOLUTE_URL_OVERRIDES () را به صورت پویا به هر مدل که در تنظیم مربوطه را برای مدل داده شده در تنظیمات برمی گرداند. ظاهر می شود اضافه می کند. این روش مربوطه را برای مدل داده شده در تنظیمات برمی گرداند.

ما آدرس کاربر user_detail را برای کاربر داده شده باز می گردیم. اکنون می توانید از get_absolute_url () در یک مثال کاربر برای بازیابی URL مربوطه استفاده کنید.

پوسته پایتون را با دستور Python management.py shell باز کنید و کد زیر را برای تست آن اجرا کنید:

```
>>> from django.contrib.auth.models import User
```

```
>>> user = User.objects.latest('id')
>>> str(user.get_absolute_url())
'/account/users/ellington/'
```

URL برگشتی همانطور که انتظار می رود ما باید نمایه ای را برای نمایه ای که تازه ساخته ایم ایجاد کنیم. فهرست و پرونده های زیر را به راهنمای برنامه حساب اضافه کنید:

```
/user/
    detail.html
    list.html
```

الگوی account/user/list.html را ویرایش کنید و کد زیر را به آن اضافه کنید

```

{% extends "base.html" %}
{% load thumbnail %}

{% block title %}People{% endblock %}

{% block content %}
<h1>People</h1>
<div id="people-list">
    {% for user in users %}
        <div class="user">
            <a href="{{ user.get_absolute_url }}>
                {% thumbnail user.profile.photo "180x180" crop="100%" as im %}
                    
                <a href="{{ user.get_absolute_url }}" class="title">
                    {{ user.get_full_name }}
                </a>
            </div>
        </div>
    {% endfor %}
</div>
{% endblock %}

```

الگوی قبلی به ما این امکان را می دهد که کلیه کاربران فعال سایت را لیست کنیم. ما نسبت به کاربران داده شده تکرار می کنیم و از تصاویر کوچک سورل استفاده می کنیم

{% thumbnail user.profile.photo "180x180" crop="100%" as im %}

الگوی پروژه خود را باز کنید و href user_listURL را در ویژگی زیر وارد کنید:

```
<li {% if section == "people" %}class="selected"{% endif %}>
  <a href="{% url "user_list" %}">People</a>
</li>
```

سرور توسعه را با دستور `python management.py runserver` شروع کرده و `http://127.0.0.1:8000/account/users` را در مرورگر خود باز کنید.

شما باید لیست از کاربران مانند موارد زیر را مشاهده کنید:

The screenshot shows a web application interface. At the top, there is a green navigation bar with the following items: 'Bookmarks' (highlighted in blue), 'My dashboard', 'Images', 'People', and 'Hello Tesla, Logout'. Below the navigation bar, the word 'People' is displayed in a large, bold, black font. Underneath this heading, there are three circular profile pictures of historical figures: Nikola Tesla, Albert Einstein, and Alan Turing. Each profile picture is accompanied by a label below it: 'Tesla' (in green), 'Einstein' (in blue), and 'Turing' (in green).

به یاد داشته باشید که اگر در ایجاد ریز عکسها مشکلی دارید، می توانید برای به دست آوردن اطلاعات اشکال در پوسته `settings.py` به پرونده `THUMBNAIL_DEBUG = True` اضافه کنید.

برنامه حساب را ویرایش کنید و کد زیر را به آن اضافه کنید:

```
{% extends "base.html" %}  
{% load thumbnail %}  
{% block title %}{{ user.get_full_name }}{% endblock %}  
{% block content %}  
<h1>{{ user.get_full_name }}</h1>  
<div class="profile-info">  
    {% thumbnail user.profile.photo "180x180" crop="100%" as im %}  
      
    {% endthumbnail %}  
</div>  
    {% with total_followers=user.followers.count %}  
        <span class="count">  
            <span class="total">{{ total_followers }}</span>  
            follower{{ total_followers|pluralize }}  
        </span>  
        <a href="#" data-id="{{ user.id }}" data-action="{% if request.user  
            in user.followers.all %}un{% endif %}follow" class="follow button">  
            {% if request.user not in user.followers.all %}  
                Follow  
            {% else %}  
                Unfollow  
            {% endif %}  
        </a>  
        <div id="image-list" class="image-container">  
            {% include "images/image/list_ajax.html" with  
                images=user.images_created.all %}  
        </div>  
    {% endwith %}  
    {% endblock %}
```

در الگوی جزئیات ، نمایه کاربر را نمایش می دهیم و از تگ قالب { % thumbnail } برای نمایش تصویر پروفایل استفاده می کنیم. ما تعداد کل دنبال کنندگان و پیوندی را برای دنبال کردن یا عدم استفاده کاربر نشان می دهیم.

درخواست AJAX را برای دنبال کردن / باز کردن یک کاربر خاص انجام می دهد. و شناسه داده ها و ویژگی های عملکرد داده را به عنصر `<a>` ، از جمله شناسه کاربر و اقدام اولیه برای انجام هنگام کلیک بر روی آن ، دنبال کنید یا آن را `unfollow` کنید ، این بستگی به این دارد که کاربر درخواست کننده صفحه دنبال کننده این کاربر دیگر باشد یا نه به عنوان مورد ممکن است.

ما تصاویر بوک مارک شده توسط کاربر را نشان می دهیم ، از جمله الگوی `images/image/list_ajax.html` دوباره مرورگر خود را باز کنید و بر روی کاربرانی که برخی از تصاویر را علامت گذاری کرده است کلیک کنید. جزئیات پروفایل را به شرح زیر مشاهده خواهید کرد:

Bookmarks

[My dashboard](#) [Images](#) [People](#)

Hello Tesla, Logout

Tesla



0 followers

FOLLOW



Django and Duke



Louis Armstrong



Chick Corea

ایجاد یک نمای AJAX برای دنبال کردن کاربران

ما یک مشاهده ساده برای follow/unfollow با استفاده از AJAX ایجاد خواهیم کرد.

پرونده views.py برنامه حساب را ویرایش کنید و کد زیر را به آن اضافه کنید:

```
from django.http import JsonResponse
from django.views.decorators.http import require_POST
from common.decorators import ajax_required
from .models import Contact

@ajax_required
@require_POST
@login_required
def user_follow(request):
    user_id = request.POST.get('id')
    action = request.POST.get('action')
    if user_id and action:
        try:
            user = User.objects.get(id=user_id)
            if action == 'follow':
                Contact.objects.get_or_create(
                    user_from=request.user,
                    user_to=user)
            else:
                Contact.objects.filter(user_from=request.user,
                                      user_to=user).delete()
            return JsonResponse({'status': 'ok'})
        except User.DoesNotExist:
            return JsonResponse({'status': 'ko'})
    return JsonResponse({'status': 'ko'})
```

نمای user_follow کاملاً شبیه به نمای تصویری است که قبلاً ایجاد کردیم. از آنجاکه ما از یک مدل واسطه سفارشی برای روابط بسیار زیاد کاربران استفاده می‌کنیم، روش‌های افزودنی پیش‌فرض و حذف مدیر خودکار از ManyToManyField در دسترس نیست.

ما از مدل Contact واسطه برای ایجاد یا حذف روابط کاربر استفاده می‌کنیم.

پرونده urls.py برنامه حساب را ویرایش کنید و الگوی URL زیر را به آن اضافه کنید:

```
path('users/follow/', views.user_follow, name='user_follow'),
```

اطمینان حاصل کنید که الگوی قبلی را قبل از الگوی URL user_detail URL قرار دهید. در غیر این صورت، هرگونه درخواست به /users/follow/ مطابقت با بیان منظم الگوی user_detail را دارد و به جای آن، این نمای اجرا خواهد شد. به یاد داشته باشید که در هر درخواست HTTP، Django آدرس درخواستی را در برابر هر الگوی به ترتیب از نظر چک می‌کند و در اولین مسابقه متوقف می‌شود.

الگوی user/detail.html برنامه حساب را ویرایش کنید و کد زیر را به آن اضافه کنید:

```
{% block domready %}

  $('a.follow').click(function(e){
    e.preventDefault();
    $.post('{% url "user_follow" %}', {
      id: $(this).data('id'),
      action: $(this).data('action')
    },
    function(data){
      if (data['status'] == 'ok') {
        var previous_action = $('a.follow').data('action');

        // toggle data-action
        $('a.follow').data('action',
          previous_action == 'follow' ? 'unfollow' : 'follow');
        // toggle link text
        $('a.follow').text(
          previous_action == 'follow' ? 'Unfollow' : 'Follow');

        // update total followers
        var previous_followers = parseInt(
          $('span.count .total').text());
        $('span.count .total').text(previous_action == 'follow' ?
          previous_followers + 1 : previous_followers - 1);
      }
    });
  });

  {% endblock %}
```

کد قبلی کد JavaScript برای انجام درخواست AJAX برای دنبال کردن یا گشودن کاربر خاص و همچنین برای پیوند follow/unfollow است. ما از jQuery برای انجام درخواست AJAX استفاده می‌کنیم و هم ویژگی داده عمل را و هم متن عنصر `a` را بر اساس مقدار قبلی آن تنظیم می‌کنیم. وقتی عمل AJAX انجام شد، ما تعداد کل دنبال کنندگان نمایش داده شده در صفحه را نیز به روز می‌کنیم. صفحه جزئیات کاربر یک کاربر موجود را باز کنید و برای آزمایش عملکردی که تازه ساخته ایم روی پیوند FOLLOW کلیک کنید. خواهید دید که تعداد پیروان افزایش می‌یابد:



ایجاد یک برنامه جریان فعالیت عمومی

بسیاری از وب سایت‌های اجتماعی یک جریان فعالیت را در اختیار کاربران خود قرار می‌دهند تا بتوانند آنچه را که کاربران دیگر روی این سیستم عامل انجام می‌دهند ردیابی کنند. یک جریان فعالیت لیستی از فعالیتهای اخیر است که توسط کاربر یا گروهی از کاربران انجام می‌شود. به عنوان مثال، خبرخوان فیس بوک یک جریان فعالیت است.

اقدامات نمونه می‌تواند کاربر X باشد. نشانگر ۷ یا کاربر X اکنون کاربر ۷ را دنبال می‌کند. ما یک برنامه کاربردی فعالیت ایجاد خواهیم کرد تا هر کاربر بتواند تعامل‌های اخیر کاربران خود را دنبال کند. برای انجام این کار، به یک مدل نیاز خواهیم داشت تا اقدامات انجام شده توسط کاربران در وب سایت را ذخیره کنیم و یک روش ساده برای افزودن اقدامات به فید.

با دستور زیر یک برنامه جدید به نام اقدامات درون پروژه خود ایجاد کنید:

```
python manage.py startapp actions
```

برای فعال کردن برنامه در پروژه خود ، برنامه جدید را به INSTALLED_APPS در پرونده setting.py پروژه خود اضافه کنید:

```
INSTALLED_APPS = [
    # ...
    'actions.apps.ActionsConfig',
]
```

پرونده model.py برنامه اعمال را ویرایش کنید و کد زیر را به آن اضافه کنید:

```
from django.db import models
class Action(models.Model):
    user = models.ForeignKey('auth.User',
        related_name='actions',
        db_index=True,
        on_delete=models.CASCADE)
    verb = models.CharField(max_length=255)
    created = models.DateTimeField(auto_now_add=True,
        db_index=True)
    class Meta:
        ordering = ('-created',)
```

کد قبلی مدل Action را نشان می دهد که برای ذخیره فعالیت های کاربر استفاده خواهد شد. زمینه های این مدل به شرح زیر است:

کاربر: کاربری که این عمل را انجام داده است. این ForeignKey به مدل کاربر جنگو است.

فعل: فعل توصیف عملی که کاربر انجام داده است.

ایجاد شده: تاریخ و زمان ایجاد این عمل. ما از auto_now_add = صحیح استفاده می کنیم تا هنگامی که جسم برای اولین بار در پایگاه داده ذخیره می شود ، این گزینه را به صورت خودکار در DateTime تنظیم کنیم.

با استفاده از این مدل اولیه ، ما فقط می توانیم اقداماتی را ذخیره کنیم ، مانند کاربر X کاری انجام داد. ما به یک زمینه خارجی اضافی احتیاج داریم تا بتوانیم اقداماتی را انجام دهیم که شامل یک هدف مورد نظر باشد ، مانند تصویر مشخص شده کاربر X یا کاربر X اکنون کاربر ۷ را دنبال می کند. همانطور که قبلًا می دانید ، یک خارجی خارجی معمولی فقط می تواند به یک مدل اشاره کند. در عوض ، ما راهی خواهیم دید که هدف هدف عمل نمونه ای از یک الگوی موجود باشد.

اینجاست که چارچوب انواع محتوای Django روی صحنه می رود.

استفاده از چارچوب انواع محتوا

Django شامل یک چارچوب نوع محتوا است که در `django.contrib.contenttypes` قرار دارد. این برنامه می تواند تمام مدل های نصب شده در پروژه شما را ردیابی کرده و یک رابط عمومی برای تعامل با مدل های شما فراهم کند.

برنامه `INSTALLED_APPS` در هنگام ایجاد یک پروژه جدید با استفاده از `django.contrib.contenttypes` انواع در تنظیم دستور پروژه شروع به طور پیش فرض درج شده است. این توسط سایر بسته های کمک کننده مانند چارچوب تأیید اعتبار و برنامه سرپرست استفاده می شود.

برنامه انواع محتوا شامل یک مدل `ContentType` است. موارد این مدل ، مدل های واقعی برنامه شما را نشان می دهد و با نصب مدل های جدید در پروژه شما ، موارد جدید `ContentType` به طور خودکار ایجاد می شوند. مدل `ContentType` دارای قسمتهای زیر است:

: این نشانگر نام برنامه ای است که مدل به آن تعلق دارد. این به طور خودکار از ویژگی `app_label` گزینه های مدل متأثر گرفته می شود. به عنوان مثال ، مدل `Image` ما به کاربرد تصاویر تعلق دارد.

: نام کلاس مدل. این نشانگر نام قابل خواندن انسان از مدل است. این به طور خودکار از ویژگی `name` گزینه های مدل `Meta` گرفته می شود. `verbose_name`

بیایید نگاهی به نحوه تعامل با اشیاء ContentType بیندازیم. پوسته را با استفاده از دستور shell python می‌توانید با انجام یک پرس و جو با ویژگی‌های app_label و ویژگی‌های مدل، management.py متناسب با یک مدل خاص را بدست آورید:

```
>>> from django.contrib.contenttypes.models import ContentType  
>>> image_type = ContentType.objects.get(app_label='images', model='image')  
>>> image_type  
<ContentType: image>
```

همچنین می‌توانید با فراخوانی متد () (کلاس) آن، کلاس مدل را از یک شی ContentType بازیابی کنید:

```
>>> image_type.model_class()  
<class 'images.models.Image'>
```

دربافت شیء ContentType برای یک کلاس مدل خاص نیز به شرح زیر معمول است:

```
>>> from images.models import Image  
>>> ContentType.objects.get_for_model(Image)  
<ContentType: image>
```

اینها فقط چند نمونه از استفاده از انواع محتوا است. جنگو راه‌های بیشتری برای همکاری با آنها ارائه می‌دهد. می‌توانید مستندات رسمی در مورد چارچوب انواع محتوا را در اینجا بیایید

[./https://docs.djangoproject.com/en/2.0/ref/contrib/contenttypes](https://docs.djangoproject.com/en/2.0/ref/contrib/contenttypes)

روابط عمومی را به مدل‌های خود اضافه کنید

در روابط عمومی ، اشیاء ContentType نقش اشاره به الگوی مورد استفاده برای روابط را ایفا می‌کنند. برای ایجاد یک رابطه عمومی در یک مدل به سه زمینه نیاز خواهید داشت:

فیلد ForeignKey به ContentType: این الگوی روابط را به ما می‌گوید

فیلد برای ذخیره کلید اصلی شی مرتبط:

این معمولاً یک PositiveIntegerField برای مطابقت با زمینه‌های اصلی کلید اصلی Django خواهد بود
زمینه‌ای برای تعریف و مدیریت ارتباط عمومی با استفاده از دو قسمت قبلی: چارچوب انواع محتوا یک زمینه GenericF
زمینه ای برای این منظور ارائه می‌دهد ForeignKey

پرونده model.py برنامه اعمال را ویرایش کنید و به صورت زیر ظاهر شوید:

```
from django.db import models
from django.contrib.contenttypes.models import ContentType
from django.contrib.contenttypes.fields import GenericForeignKey

class Action(models.Model):
    user = models.ForeignKey('auth.User',
                           related_name='actions',
                           db_index=True,
                           on_delete=models.CASCADE)
    verb = models.CharField(max_length=255)
    target_ct = models.ForeignKey(ContentType,
                                 blank=True,
```

```

        null=True,
        related_name='target_obj',
        on_delete=models.CASCADE)
target_id = models.PositiveIntegerField(null=True,
                                         blank=True,
                                         db_index=True)
target = GenericForeignKey('target_ct', 'target_id')
created = models.DateTimeField(auto_now_add=True,
                               db_index=True)

class Meta:
    ordering = ('-created',)

```

ما قسمتهای زیر را به مدل Action اضافه کرده ایم:

فیلد ForeignKey که به مدل ContentType اشاره دارد: target_ct

یک PositiveIntegerField برای ذخیره کردن کلید اصلی شی مرتبط: target_id

یک فیلد GenericForeignKey با استفاده از دو قسمت قبلی، به موضوع مرتبط با آن مربوط می شود:

Django هیچ زمینه ای در پایگاه داده برای زمینه های GenericForeignKey ایجاد نمی کند. تنها فیلدهایی که در زمینه های پایگاه داده نقشه برداری می شوند، target_ct و target_id هستند. هر دو قسمت دارای `blank=True` و صفت `null=True` هستند به طوری که در هنگام `save` در اشیاء عملی، یک هدف مورد نیاز نمی باشد.

برای ایجاد مهاجرت اولیه برای این برنامه ، دستور زیر را اجرا کنید:

```
python manage.py makemigrations actions
```

باید خروجی زیر را مشاهده کنید:

```
Migrations for 'actions':
```

```
  | actions/migrations/0001_initial.py
  |     - Create model Action
```

سپس دستور بعدی را برای همگام سازی برنامه با بانک اطلاعات اجرا کنید:

```
python manage.py migrate
```

خروجی فرمان باید نشان دهد که مهاجرت های جدید به شرح زیر اعمال شده است:

```
Applying actions.0001_initial... OK
```

باید مدل Action را به سایت `administrat` اضافه کنیم. `admin.py` را ویرایش کنید

پرونده برنامه اقدامات را وارد کنید و کد زیر را به آن اضافه کنید:

```
from django.contrib import admin
from .models import Action

@admin.register(Action)
class ActionAdmin(admin.ModelAdmin):
    list_display = ('user', 'verb', 'target', 'created')
    list_filter = ('created',)
    search_fields = ('verb',)
```

شما فقط مدل Action را در سایت مدیریت ثبت کرده اید. دستور اولیه سرور python management.py را اجرا کنید تا سرور توسعه را تنظیم کنید و <http://127.0.0.1:8000/admin/actions/action/add> را در مرورگر خود باز کنید.
برای ایجاد یک شی جدید از Action باید صفحه را به شرح زیر مشاهده کنید:

The screenshot shows the Django Admin interface for adding a new Action. The top navigation bar displays 'Django administration' on the left and 'WELCOME, ANTONIO. VIEW SITE / CHANGE PASSWORD / LOG OUT' on the right. Below the navigation, a breadcrumb trail shows 'Home > Actions > Actions > Add action'. The main title 'Add action' is centered above the form fields. There are four input fields: 'User' (a dropdown menu with a '+' icon), 'Target ct' (another dropdown menu), 'Target id' (a dropdown menu), and 'Verb' (a text input field). At the bottom right, there are three buttons: 'Save and add another', 'Save and continue editing', and a larger 'SAVE' button.

Django administration

WELCOME, ANTONIO. VIEW SITE / CHANGE PASSWORD / LOG OUT

Home > Actions > Actions > Add action

Add action

User:

Target ct:

Target id:

Verb:

همانطور که در عکس قبلی مشاهده می کنید ، فقط قسمتهای target_ct و target_id که در زمینه های پایگاه داده واقعی نقشه برداری شده اند نشان داده شده است. قسمت GenericForeignKey به شکل ظاهر نمی شود. قسمت target_ct به شما امکان می دهد تا هر یک از مدلهای ثبت شده پروژه Django خود را انتخاب کنید. شما می توانید انواع target_ct محتوا را انتخاب کنید تا از میان مجموعه محدودی از مدلها با استفاده از ویژگی limit_choices_to در قسمت target_ct محدودیت ها را انتخاب کنید: ویژگی limit_choices_to به شما امکان می دهد محتوای زمینه های ForeignKey را به مجموعه ای خاص از مقادیر محدود کنید.

یک پرونده جدید را درون فهرست برنامه کاربردی ایجاد کنید و نام آن را utils.py بگذارید. ما یک کارکرد میانبر را تعریف خواهیم کرد که به ما امکان می دهد اشیاء جدید Action را به روشی ساده ایجاد کنیم. پرونده utils.py جدید را ویرایش کنید و کد زیر را به آن اضافه کنید:

```
from django.contrib.contenttypes.models import ContentType
from .models import Action
```

```
def create_action(user, verb, target=None):
    action = Action(user=user, verb=verb, target=target)
    action.save()
```

تابع `create_action()` به ما امکان می دهد اقداماتی ایجاد کنیم که به صورت اختیاری یک هدف را شامل می شود. ما می توانیم از این عملکرد در هر کجای کد خود به عنوان میانبر استفاده کنیم تا اقدامات جدیدی به جریان فعالیت اضافه شود.

اجتناب از اقدامات تکراری در جریان فعالیت

بعضی اوقات ، کاربران شما ممکن است چندین بار یک عمل را انجام دهند.

آنها ممکن است چندین بار بر روی دکمه های LIKE یا UNLIKE کلیک کرده و یا در یک بازه زمانی کوتاه چندین بار همین کار را انجام دهند.

این به راحتی منجر به ذخیره و نمایش اقدامات تکراری خواهد شد. برای جلوگیری از این ، ما تابع `create_action()` را برای پریش از اقدامات تکراری واضح بهبود خواهیم داد.

پرونده `utils.py` برنامه اقدامات را به شرح زیر ویرایش کنید:

```

import datetime
from django.utils import timezone
from django.contrib.contenttypes.models import ContentType
from .models import Action

def create_action(user, verb, target=None):
    # check for any similar action made in the last minute
    now = timezone.now()
    last_minute = now - datetime.timedelta(seconds=60)
    similar_actions = Action.objects.filter(user_id=user.id,
                                             verb=verb,
                                             created__gte=last_minute)
    if target:
        target_ct = ContentType.objects.get_for_model(target)
        similar_actions = similar_actions.filter(
            target_ct=target_ct,
            target_id=target.id)
    if not similar_actions:
        # no existing actions found
        action = Action(user=user, verb=verb, target=target)
        action.save()
        return True
    return False

```

ما برای جلوگیری از وجود اقدامات تکراری و بازگشت Boolean ، عملکرد "ایجاد (علامت)" را تغییر داده ایم تا بگوییم این عمل ذخیره شده است یا خیر. اینگونه است که از تکرارها جلوگیری می کنیم:

ابتدا زمان فعلی را با استفاده از متود `timezone.now()` دریافت می کنیم. این روش مانند `USE_TZ=True` عمل می کند اما یک شی آگاه از منطقه را برای گرداند. Django `DateTime.datetime.now` تنظیماتی به نام `settings.py` ایجاد شده با استفاده از برای فعال یا غیرفعال کردن پشتیبانی از منطقه زمانی ارائه می دهد. پرونده پیش فرض `USE_TZ=True` دستور پروژه شروع شامل است.

ما از متغیر last_minute برای ذخیره DateTime از یک دقیقه قبل استفاده می کنیم و اقدامات مشابهی را که از آن زمان توسط کاربر انجام می شود بازیابی می کنیم.

اگر در همان لحظه آخر هیچ اقدام مشابهی وجود نداشته باشد ، یک شیء Action ایجاد می کنیم. اگر یک شیء Action شد ، در غیر اینصورت False ، ما به حالت True باز می گردیم.

افزودن اقدامات کاربر به جریان فعالیت

وقت آن است که برای ایجاد جریان فعالیت برای کاربران خود ، برخی از اقدامات را به نظرات خود اضافه کنیم. ما یک عمل برای هر یک از تعامل های زیر ذخیره خواهیم کرد:

یک کاربر یک عکس را علامت گذاری می کند

یک کاربر یک تصویر را دوست دارد

یک کاربر یک حساب کاربری ایجاد می کند

یک کاربر به دنبال یک کاربر دیگر شروع می کند

فایل views.py برنامه تصاویر را ویرایش کرده و واردات زیر را اضافه کنید:

```
from actions.utils import create_action
```

در نمای image_create ، بعد از ذخیره کردن تصویر ، مانند ایجاد این گزینه:

```
new_item.save()
create_action(request.user, 'bookmarked image', new_item)
```

در نمای image_like ، بعد از افزودن کاربر به رابطه user_like را به شرح زیر اضافه کنید:

```
image.users_like.add(request.user)
create_action(request.user, 'likes', image)
```

اکنون پرونده views.py برنامه حساب را ویرایش کرده و واردات زیر را اضافه کنید:

```
from actions.utils import create_action
```

در نمای ثبت نام ، create_action() را بعد از ایجاد شیء Profile به شرح زیر اضافه کنید:

```
Profile.objects.create(user=new_user)
create_action(new_user, 'has created an account')
```

در نمای create_action() ، user_follow را اضافه کنید:

```
Contact.objects.get_or_create(user_from=request.user,
                                user_to=user)
create_action(request.user, 'is following', user)
```

همانطور که در کد قبلی مشاهده می کنید ، به لطف مدل Action و عملکرد یاور ما ، save در اقدامات جدید در جریان فعالیت بسیار آسان است.

نمایش جریان فعالیت(Displaying the activity stream)

سرانجام به روشی برای نمایش جریان فعالیت برای هر کاربر نیاز خواهیم داشت. ما جریان فعالیت را در داشبورد کاربر قرار خواهیم داد.

پرونده views.py برنامه حساب را ویرایش کنید. مدل Action را وارد کنید و نمای داشبورد را به شرح زیر تغییر دهید:

```
from actions.models import Action

@login_required
def dashboard(request):
    # Display all actions by default
    actions = Action.objects.exclude(user=request.user)
    following_ids = request.user.following.values_list('id',
                                                       flat=True)
    if following_ids:
        # If user is following others, retrieve only their actions
        actions = actions.filter(user_id__in=following_ids)
    actions = actions[:10]

    return render(request,
                  'account/dashboard.html',
                  {'section': 'dashboard',
                   'actions': actions})
```

در نمای قبلی ، ما تمام اقدامات را از پایگاه داده بازیابی می کنیم ، به استثنای موارد انجام شده توسط کاربر فعلی. به طور پیش فرض ، ما آخرین اقدامات انجام شده توسط همه کاربران در سیستم عامل را بازیابی خواهیم کرد. اگر کاربر دنبال کننده سایر کاربران است ، ما درخواست را محدود می کنیم تا فقط اقدامات انجام شده توسط کاربران که دنبال می کنند ، بازیابی شود.

سرانجام ، نتیجه را به 10 عمل اول بازگشتی محدود می کنیم. ما از `QuerySet order_by()` استفاده نمی کنیم زیرا به سفارش پیش فرض ما در گزینه های متن مدل اکشن تکیه می کنیم. اقدامات اخیر اولین بار خواهد بود زیرا ما در مدل `Action` `ordering = ('-created')` را تنظیم کرده ایم.

بهینه سازی QuerySets که شامل اشیاء مرتبط می باشد

هر بار که یک شیء `Action` را بازیابی می کنید ، معمولاً به موضوع کاربر مرتبط با آن و شیء `Profile` مربوط به کاربر دسترسی خواهید داشت. Django ORM یک روش ساده برای بازیابی اشیاء مرتبط با هم زمان ارائه می دهد و از این طریق از پرس و جوهای اضافی به پایگاه داده اجتناب می کند.

استفاده از `select_related()`

Django روشنی `QuerySet` به نام `select_related()` را ارائه می دهد که به شما امکان می دهد اشیاء مرتبط را برای روابط یک به چند بازیابی کنید. این ترجمه به یک `QuerySet` پیچیده تر و پیچیده تبدیل می شود ، اما هنگام دسترسی به اشیاء مرتبط از پرس و جوهای اضافی جلوگیری می کنید. روش `select_related()` برای قسمت های `OneToOne` و `ForeignKey` است.

این کار با انجام یک SQL JOIN و شامل زمینه های موضوع مرتبط در جمله `SELECT` انجام می شود.

برای استفاده از `Select_related()` ، خط زیر کد قبلی را ویرایش کنید:

```
actions = actions[:10]
```

همچنین ، زمینه های انتخابی خود را مانند این انتخاب کنید:

```
actions = actions.select_related('user', 'user__profile')[:10]
```

ما برای پیوستن به جدول پروفایل در یک پرس و جو SQL واحد ، از user__profile استفاده می کنیم. اگر بدون انتخاب هیچگونه استدلالی به آن ، با Select_related تماس بگیرید ، آن را از همه روابط ForeignKey بازیابی می کند. همیشه Select_related را به روابطی که پس از آن دسترسی خواهید داشت محدود کنید.

با استفاده از prefetch_related

Select_related به شما در افزایش عملکرد برای بازیابی اشیاء مرتبط در روابط یک به چند کمک می کند. با این حال ، Select_related نمی تواند برای روابط بسیاری به بسیاری یا خیلی به یک (زمینه های ManyToMany) یا معکوس (ForeignKey) کار کند. Django متد QuerySet متفاوتی به نام prefetch_related ارائه می دهد که علاوه بر روابط select-related روش prefetch_related یک جستجوی جداگانه برای هر رابطه نیز انجام می دهد و با پشتیبانی شده با استفاده از نتایج پیوسته می شود.

این روش همچنین از پیش تنظیم از GenericRelation و ForeignKey پشتیبانی می کند. پرونده views.py حساب برنامه را ویرایش کنید و با افزودن prefetch_related به آن برای قسمت مورد نظر target پرونده خود را تکمیل کنید ، به شرح زیر:

```
actions = actions.select_related('user', 'user__profile')\n        .prefetch_related('target')[:10]
```

این پرس و جو اکنون برای بازیابی اقدامات کاربر از جمله اشیاء مرتبط بهینه شده است.

Create Template for action

اکنون الگوی ایجاد شده برای نمایش یک شیء خاص Action را ایجاد خواهیم کرد. دایرکتوری جدیدی را درون فهرست برنامه کاربردی ایجاد کنید و از آن الگو بگیرید. ساختار پرونده زیر را به آن اضافه کنید:

```
actions/
  action/
    detail.html
```

پرونده الگوی action / action / detail.html را ویرایش کنید و خطوط زیر را به آن اضافه کنید:

```
{% load thumbnail %}

{% with user=action.user profile=action.user.profile %}
<div class="action">
  <div class="images">
    {% if profile.photo %}
      {% thumbnail user.profile.photo "80x80" crop="100%" as im %}
      <a href="{{ user.get_absolute_url }}>
        
      </a>
    {% endthumbnail %}
    {% endif %}

    {% if action.target %}
      {% with target=action.target %}
        {% if target.image %}
          {% thumbnail target.image "80x80" crop="100%" as im %}
          <a href="{{ target.get_absolute_url }}>
            
          </a>
        {% endthumbnail %}
        {% endif %}
      {% endwith %}
    {% endif %}
  </div>
```

```

<div class="info">
  <p>
    <span class="date">{{ action.created|timesince }} ago</span>
    <br />
    <a href="{{ user.get_absolute_url }}">
      {{ user.first_name }}
    </a>
    {{ action.verb }}
    {% if action.target %}
      {% with target=action.target %}
        <a href="{{ target.get_absolute_url }}">{{ target }}</a>
      {% endwith %}
    {% endif %}
  </p>
</div>
</div>
{% endwith %}

```

این الگوی است که برای نمایش یک شیء Action استفاده می شود. ابتدا ما از { % with } کنیم تا کاربر در حال انجام عمل و شیء مربوط به Profile بازیابی شود. سپس اگر شیء Action دارای یک هدف مرتبط باشد ، تصویر هدف مورد نظر را نمایش می دهیم. سرانجام ، در صورت وجود ، پیوند را به کاربرانی که عمل ، فعل و هدف را انجام داده اند ، نمایش می دهیم.

اکنون الگوی حساب / dashboard.html برنامه حساب را ویرایش کرده و کد زیر را در قسمت انتهای بخش محتوا اضافه کنید:

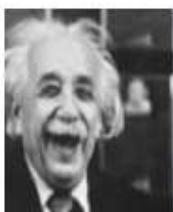
```

<h2>What's happening</h2>
<div id="action-list">
  {% for action in actions %}
    {% include "actions/action/detail.html" %}
  {% endfor %}
</div>

```

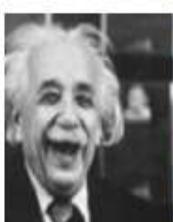
/ را در مرورگر خود باز کنید. با یک کاربر موجود وارد شوید و چندین کار را انجام دهید تا آنها در پایگاه داده ذخیره شوند. سپس با استفاده از یک کاربر دیگر وارد سیستم شوید ، کاربر قبلی را دنبال کنید و به صفحه عملکرد اقدام شده در صفحه داشبورد نگاهی بیندازید. باید مانند زیر باشد:

What's happening



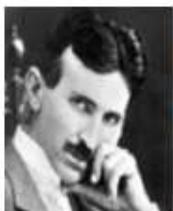
3 minutes ago

Einstein likes Alternating electric current generator



5 minutes ago

Einstein bookmarked image Turing Machine



2 days, 2 hours ago

Tesla likes Chick Corea

ما فقط یک فعالیت کامل برای کاربران خود ایجاد کرده ایم و به راحتی می توانیم اقدامات جدید کاربران را به آن اضافه کنیم. همچنین می توانید با اجرای همان صفحه اصلی AJAX که برای مشاهده image_list استفاده کرده اید ، قابلیت پیمایش نامحدود را به جریان فعالیت اضافه کنید.

استفاده از سیگنال ها برای شمارش آزار دهنده تعداد

مواردی وجود دارد که شما می خواهید داده های خود را غیر اخلاقی کنید. Denormalization باعث می شود که داده ها به گونه ای نابود شوند که عملکرد خواندن را بهینه می کند. شما باید در مورد ناهنجاری سازی مراقب باشید و هنگامی که واقعاً به آن احتیاج دارید ، از آن استفاده کنید. بزرگترین مسئله ای که با تغییر شکل پیدا خواهید کرد اینست که به روز کردن داده های غیرمتربقه خود دشوار است.

ما به نمونه ای از چگونگی بهبود جستجوی خود با شمارش غیرقانونی نگاه خواهیم کرد. اشکال این است که ما باید داده های اضافی را به روز کنیم. ما داده ها را از مدل Image خود تغییر داده و از سیگنال های جنگو برای به روزرسانی اطلاعات استفاده خواهیم کرد.

کار با سیگنال ها

جنگو با یک دستگاه پخش کننده سیگنال ارائه می شود که به توابع گیرنده اجازه می دهد تا هنگام انجام برخی اقدامات خاص ، از آنها مطلع شوند. هر وقت اتفاق دیگری رخ می دهد ، سیگنال ها بسیار مفید هستند. همچنین می توانید سیگنال های خود را بسازید تا دیگران هنگام وقوع یک رویداد از آن مطلع شوند.

Django سیگنال های مختلفی را برای مدل های واقع در `django.db.models.signals` فراهم می کند. برخی از این سیگنال ها به شرح زیر است:

`post_save` و `pre_save` قبل یا بعد از فراخوانی با روش ذخیره یک مدل ارسال می شوند

وقتی یک ManyToManyField روی یک مدل تغییر می کند ، m2m_changed ارسال می شود

اینها فقط زیر مجموعه‌ای از سیگنال‌های ارائه شده توسط جنگو است. می‌توانید لیست تمام سیگنال‌های داخلی را در <https://docs.djangoproject.com/en/2.0/ref/signals> پیدا کنید.

بایایید بگوییم که می خواهید تصاویر را با محبوبیت بازیابی کنید. برای بازیابی تصاویر سفارش داده شده توسط تعدادی از کاربرانی که آنها را دوست دارند ، می توانید از توابع جمع Django استفاده کنید. به یاد داشته باشید که توابع جمع آوری جنگو را در بخش 3 ، گسترش برنامه و بلاگ خود استفاده کرده اید.

کد زیر تصاویر را با توجه به تعداد لایک آنها بازیابی می کند:

```
from django.db.models import Count  
from images.models import Image
```

```
images_by_popularity = Image.objects.annotate(
    total_likes=Count('users_like')).order_by('-total_likes')
```

با این حال ، سفارش تصاویر با شمارش تعداد لایک آنها از لحاظ عملکرد گرانتر از سفارش آنها توسط یک فیلد است که تعداد کل آن را ذخیره می کند. شما می توانید فیلدی به مدل تصویر اضافه کنید تا تعداد لایک ها برای افزایش کارایی در نمایش داده ها، که شامل این نوبته هستند، غیرفعال شود. حال، مسئله این است که چگونه عرضه این قسمت را به بروز کرد؟

ب و ن د ه `model.py` استفاده از تصاویر را و باش، کنید و قسمت `total_likes` را به مدل تصویر اضافه کنید:

قسمت total_likes به ما امکان می دهد تعداد کل کاربرانی که مانند هر تصویر را دوست دارند را ذخیره کنیم. وقتی می خواهید QuerySets آنها بخرید و یا سفارش دهید تعداد Denormalizing کردن مفید است.

برای ایجاد مهاجرت برای اضافه کردن فیلد جدید به جدول پایگاه داده ، دستور زیر را اجرا کنید:

```
python manage.py makemigrations images
```

باید خروجی زیر را مشاهده کنید:

```
Migrations for 'images':  
  images/migrations/0002_image_total_likes.py  
    - Add field total_likes to image
```

سپس دستور زیر را برای اعمال مهاجرت اجرا کنید:

```
python manage.py migrate images
```

خروجی باید شامل خط زیر باشد:

```
Applying images.0002_image_total_likes... OK
```

ما یک عملکرد گیرنده را به سیگنال m2m_changed وصل خواهیم کرد. یک پرونده جدید را درون فهرست برنامه کاربردی تصاویر ایجاد کنید و آن را signals.py بنویسید.

کد زیر را به آن اضافه کنید:

```
from django.db.models.signals import m2m_changed
from django.dispatch import receiver
from .models import Image

@receiver(m2m_changed, sender=Image.users_like.through)
def users_like_changed(sender, instance, **kwargs):
    instance.total_likes = instance.users_like.count()
    instance.save()
```

ابتدا تابع user_like_changed را به عنوان تابع گیرنده با استفاده از دکوراسیون گیرنده () ثبت می کنیم و آن را به سیگنال m2m_changed وصل می کنیم.

ما عملکرد را به Image.users_like.through متصل می کنیم تا عملکرد تنها در صورتی فراخوانی شود که سیگنال m2m_changed توسط این فرستنده راه اندازی شده باشد. یک روش جایگزین برای ثبت یک عملکرد گیرنده وجود دارد که شامل استفاده از روش اتصال () شیء Signal است.

سیگنال های جنگو همزمان و مسدود کننده هستند. سیگنال ها را با کارهای ناهمزمان اشتباه نگیرید. با این حال ، هنگامی که کد شما توسط یک سیگنال اطلاع می یابد ، می توانید هر دو را برای راه اندازی کارهای غیرهمسان ترکیب کنید.

شما باید عملکرد گیرنده خود را به یک سیگنال متصل کنید تا هر بار ارسال سیگنال صدا شود. روش توصیه شده برای ثبت سیگنال های شما با وارد کردن آنها به روش () ready کلاس پیکربندی برنامه شما است. Django یک رجیستری برنامه را فراهم می کند که به شما امکان می دهد برنامه های خود را پیکربندی و بررسی کنید.

کلاس پیکربندی برنامه

Django به شما امکان می دهد کلاس های پیکربندی را برای برنامه های خود تعیین کنید. وقتی یک برنامه با استفاده از دستور startapp ایجاد می کنید ، Django یک پرونده apps.py را به فهرست برنامه ها اضافه می کند ، از جمله تنظیمات اولیه برنامه که از کلاس AppConfig به ارت می برد.

کلاس پیکربندی برنامه به شما امکان ذخیره ابرداده و پیکربندی برنامه را می دهد و درون برنامه ریزی را برای برنامه فراهم می کند. می توانید اطلاعات بیشتر در مورد تنظیمات برنامه را در <https://docs.djangoproject.com/fa/2.0/ref/applications> پیدا کنید.

به منظور ثبت توابع گیرنده سیگنال خود ، هنگام استفاده از دکوراتور() resiver ، فقط باید مازول سیگنال برنامه خود را درون روش ready() کلاس پیکربندی برنامه وارد کنید. این روش به محض جمع شدن کامل رجیستری برنامه فراخوانی می شود. هر نوع ابتکار دیگری برای درخواست شما نیز باید در این روش گنجانده شود.

پرونده apps.py تصاویر را ویرایش کنید و به صورت زیر ظاهر کنید:

```
from django.apps import AppConfig

class ImagesConfig(AppConfig):
    name = 'images'

    def ready(self):
        # import signal handlers
        import images.signals
```

ما سیگنال های این برنامه را به روش ready() آماده می کنیم تا هنگام بارگذاری برنامه تصاویر ، آنها وارد شوند.

سرور توسعه را با دستور زیر اجرا کنید:

```
python manage.py runserver
```

مرورگر خود را باز کنید تا یک صفحه جزئیات تصویر را ببینید و بر روی دکمه LIKE کلیک کنید. به سایت مدیریت برگردید ، به URL ویرایش تصویر مانند <http://127.0.0.1:8000/admin/images/image/1/change> بروید و به ویژگی total_likes نگاهی بیندازید. باید ببینید که ویژگی total_likes با تعداد کل کاربرانی که تصویر را دوست دارند ، به شرح زیر است:

Users like:



Hold down "Control", or "Command" on a Mac, to select more than one.

Total likes:



اکنون می توانید از صفات `total_likes` برای سفارش تصاویر با محبوبیت استفاده کنید یا مقدار را در هر نقطه نمایش دهید ، از پرس و جوهای پیچیده برای محاسبه آن اجتناب کنید. برای گرفتن تصاویر مرتب شده با توجه به تعداد مشابه آنها ، عبارت زیر را در نظر بگیرید:

```
from django.db.models import Count

images_by_popularity = Image.objects.annotate(
    likes=Count('users_like')).order_by('-likes')
```

جستجوی قبلی اکنون می تواند به شرح زیر نوشته شود:

```
images_by_popularity = Image.objects.order_by('-total_likes')
```

این منجر به پرس و جو SQL ارزانتر می شود. این فقط نمونه ای از نحوه استفاده از سیگنال های جنگو است.

با احتیاط از سیگنال ها استفاده کنید زیرا آنها شناخت جریان کنترل را دشوار می کنند.

در بسیاری از موارد ، اگر می دانید کدام گیرنده باید به آنها اطلاع داده شود ، می توانید از استفاده از سیگنال ها خودداری کنید. برای مطابقت با وضعیت فعلی پایگاه داده ، باید تعداد اولیه را تنظیم کنید. پوسته را با دستور python management.py shell باز کنید و کد زیر را اجرا کنید:

```
from images.models import Image
for image in Image.objects.all():
    image.total_likes = image.users_like.count()
    image.save()
```

تعداد لایک ها برای هر تصویر اکنون به روز است.

استفاده از Redis برای ذخیره نماهای مورد

Redis یک پایگاه داده کلید / ارزش پیشرفته است که به شما امکان می دهد انواع مختلفی از داده ها را ذخیره کنید و در عملیات O / 1 بسیار سریع است. Redis همه چیز را در حافظه ذخیره می کند ، اما داده ها می توانند با رهایت داده به دیسک هر بار یکبار یا اضافه کردن هر دستور به یک سیاهه ، داده ها ادامه یابد. Redis در مقایسه با فروشگاه های مهم / کلید دیگر بسیار متنوع است: مجموعه ای از دستورات قدرتمند را فراهم می کند و از ساختار داده های متنوعی مانند رشته ها ، هش ها ، لیست ها ، مجموعه ها ، مجموعه های مرتب شده و حتی bitmaps HyperLogLogs یا پشتیبانی می کند.

اگرچه SQL به بهترین وجه برای ذخیره سازی داده های مداوم تعریف شده از طرحواره مناسب است ، Redis هنگام برخورد با تغییر سریع داده ها ، ذخیره سازی فرار یا هنگامی که به یک ذخیره سریع نیاز دارید ، مزایای بی شماری را ارائه می دهد. بایاید نگاهی بیندازیم که چگونه می توان از Redis برای ایجاد قابلیت های جدید در پروژه ما استفاده کرد.

نصب Redis

آخرین نسخه Redis را از <https://redis.io/download> بارگیری کنید. فایل tar.gz را از حالت فشرده خارج کنید ، وارد فهرست تغییر فهرست شوید و با استفاده از دستور make ، Redis را کامپایل کنید:

```
cd redis-4.0.9  
make
```

پس از نصب آن ، از دستور shell زیر برای تنظیم اولیه سرور Redis استفاده کنید:

```
src/redis-server
```

باید خروجی را ببینید که با خطوط زیر پایان می یابد:

```
# Server initialized  
* Ready to accept connections
```

به طور پیش فرض ، Redis روی پورت 6379 کار می کند. شما می توانید با استفاده از پرچم --port ، به عنوان مثال ، سرور مجدد 6655 --port دلخواه را مشخص کنید.

سرور Redis را نگه دارید و پوسته دیگری را باز کنید. مشتری Redis را با دستور زیر شروع کنید:

```
src/redis-cli
```

پوسته مشتری Redis مانند این را می بینید:

```
127.0.0.1:6379>
```

مشتری Redis به شما امکان می دهد دستورات Redis را مستقیماً از پوسته اجرا کنید. بباید برخی دستورات را امتحان کنیم. برای ذخیره یک مقدار در کلید ، دستور SET را در پوسته Redis وارد کنید:

```
127.0.0.1:6379> SET name "Peter"
OK
```

دستور قبلی یک کلید نام با مقدار رشته "پیتر" در پایگاه داده Redis ایجاد می کند. خروجی OK نشان می دهد که کلید با موفقیت ذخیره شده است. سپس مقدار را با استفاده از دستور GET بازیابی کنید ، به شرح زیر:

```
127.0.0.1:6379> GET name
"Peter"
```

همچنین می توانید با استفاده از دستور EXISTS یک کلید وجود داشته باشد یا خیر. در صورت وجود کلید داده شده ، این دستور 1 بر می گردد ، در غیر این صورت:

```
127.0.0.1:6379> EXISTS name
(integer) 1
```

می توانید با استفاده از دستور EXPIRE زمان انقضای یک کلید را تنظیم کنید ، که به شما امکان می دهد زمان زندگی برای چند ثانیه را تعیین کنید. گزینه دیگر استفاده از دستور EXPIREAT است که انتظار یک تایم یونیکس را دارد. انقضای کلید برای استفاده از Redis به عنوان حافظه پنهان یا ذخیره داده های فرار مفید است:

```
127.0.0.1:6379> GET name
"Peter"
127.0.0.1:6379> EXPIRE name 2
(integer) 1
```

دو ثانیه صبر کنید و سعی کنید دوباره همان کلید را بدست آورید:

```
127.0.0.1:6379> GET name  
(nil)
```

پاسخ (صفر) یک پاسخ صفر است و به این معنی است که هیچ کلید پیدا نشده است. همچنین می توانید با استفاده از دستور DEL هر کلید را به شرح زیر حذف کنید:

```
127.0.0.1:6379> SET total 1  
OK  
127.0.0.1:6379> DEL total  
(integer) 1  
127.0.0.1:6379> GET total  
(nil)
```

اینها فقط دستورات اساسی برای عملیات کلیدی هستند. Redis شامل مجموعه بزرگی از دستورات برای انواع دیگر داده ها، مانند رشته ها، هش ها، مجموعه ها و مجموعه های مرتب شده است. می توانید به کلیه دستورات Redis در <https://redis.io/topics/data-types> و کلیه انواع داده Redis در <https://redis.io/commands> نگاهی بیندازید.

استفاده از Redis با پایتون

ما به پیوند پایتون برای Redis نیاز خواهیم داشت. با استفاده از دستور زیر، redis-py را از طریق پیپ نصب کنید:

```
pip install redis==2.10.6
```

می توانید اسناد redis-py را در <https://redis-py.readthedocs.io> پیدا کنید.

بسته redis-py دو کلاس برای تعامل با Redis ارائه می دهد:

Redis و StrictRedis هر دو عملکرد مشابه را ارائه می دهند. کلاس StrictRedis تلاش می کند تا به نحو فرمان رسمی Redis پایبند باشد. کلاس StrictRedis را گسترش می دهد و برخی از روش ها را برای ارائه سازگاری به عقب رد می کند. ما از StrictRedis استفاده خواهیم کرد زیرا از نحوی دستور Redis پیروی می کند. پوسته پایتون را باز کرده و کد زیر را اجرا کنید:

```
>>> import redis  
>>> r = redis.StrictRedis(host='localhost', port=6379, db=0)
```

کد قبلی ارتباطی با بانک اطلاعاتی Redis ایجاد می کند. در Redis ، بانکهای اطلاعاتی به جای نام بانک اطلاعاتی توسط یک فهرست عدد صحیح شناسایی می شوند. به طور پیش فرض ، مشتری به پایگاه داده 0 وصل می شود.

تعداد دیتابیس های موجود در Redis به 16 تنظیم شده است ، اما می توانید این مورد را در پرونده پیکربندی redis.conf تغییر دهید.

اکنون با استفاده از پوسته پایتون یک کلید تنظیم کنید:

```
>>> r.set('foo', 'bar')  
True
```

این دستور درست است ، نشان می دهد که کلید بوده است

با موفقیت ایجاد شد اکنون می توانید با استفاده از دستور get کلید را بازیابی کنید:

```
>>> r.get('foo')  
b'bar'
```

همانطور که از کد قبلی می توانید توجه کنید ، روش های StrictRedis از نحو دستور Redis پیروی می کنند.

بیایید Redis را در پروژه خود ادغام کنیم. فایل setting.py پروژه بوکمارک ها را ویرایش کنید و تنظیمات زیر را به آن اضافه کنید:

```
REDIS_HOST = 'localhost'  
REDIS_PORT = 6379  
REDIS_DB = 0
```

این تنظیمات مربوط به سرور Redis و بانک اطلاعاتی است که ما برای پروژه خود استفاده خواهیم کرد.

ذخیره نماهای مورد در Redis

باید راهی برای ذخیره تعداد کل دفعات مشاهده شده از تصویر بیابیم. اگر این کار را با استفاده از Django ORM پیاده سازی کنیم ، هر بار که نمایش داده می شود ، یک پرس و جو به روزرسانی SQL را شامل می شود. اگر در عوض از Redis استفاده کنیم ، فقط کافی است یک پیشخوان ذخیره شده در حافظه را افزایش دهیم ، که منجر به عملکرد بسیار بهتر و سریار کمتر می شود.

فایل views.py برنامه تصاویر را ویرایش کرده و پس از اظهارات واردات موجود ، کد زیر را به آن اضافه کنید:

```
import redis  
from django.conf import settings  
  
# connect to redis  
r = redis.StrictRedis(host=settings.REDIS_HOST,  
                      port=settings.REDIS_PORT,  
                      db=settings.REDIS_DB)
```

با کد قبلی اتصال Redis را ایجاد می کنیم تا از آن در نظرات خود استفاده کنیم. نمای image_detail را ویرایش کرده و به صورت زیر ظاهر کنید:

```
def image_detail(request, id, slug):
    image = get_object_or_404(Image, id=id, slug=slug)
    # increment total image views by 1
    total_views = r.incr('image:{}:views'.format(image.id))
    return render(request,
                  'images/image/detail.html',
                  {'section': 'images',
                   'image': image,
                   'total_views': total_views})
```

در این نمای ، از دستور incr استفاده می کنیم که مقدار a را افزایش می دهد

اگر کلید وجود ندارد ، دستور incr قبل آن را ایجاد می کند. روش incr () مقدار نهایی کلید را بعد از انجام عمل برمی گرداند. ما مقدار را در متغیر total_views ذخیره می کنیم و آن را در متن قالب می گذرانیم. ما کلید Redis را با استفاده از یک نماد ، مانند نوع اشیاء: id: فیلد می سازیم (برای مثال ، تصویر: 33:id).

کنوانسیون نامگذاری کلیدهای Redis استفاده از علامت کولون به عنوان جداکننده برای ایجاد کلیدهای دارای نام است. با انجام این کار ، نامهای کلیدی به خصوص کلامی هستند و کلیدهای مرتبط بخشی از همان طرح را در نام خود به اشتراک می گذارند.

بعد از عنصر <الگوی images / image / detail.html> استفاده از تصویر را ویرایش کنید و کد زیر را به آن اضافه کنید:

```
<span class="count">
  {{ total_views }} view{{ total_views|pluralize }}
</span>
```

اکنون یک صفحه جزئیات تصویر را در مرورگر خود باز کرده و چندین بار بارگذاری مجدد کنید. خواهید دید که هر بار مشاهده پردازش می شود ، کل نمایشهای نمایش داده شده توسط 1. افزایش می یابد. به مثال زیر توجه کنید:

Django and Duke



0 likes

16 views

LIKE

Django and Duke image.

Nobody likes this image yet.

عالی! شما با موفقیت Redis را در پروژه خود ذخیره کرده اید تا تعداد موارد را ذخیره کنید.

ذخیره رتبه بندی در Redis

بایاید چیزی پیچیده تر با Redis بسازیم. ما یک رتبه بندی از تصاویر مشاهده شده بر روی پلتفرم خود ایجاد خواهیم کرد. برای ساختن این رده بندی از مجموعه های مرتب شده Redis استفاده خواهیم کرد. مجموعه مرتب شده مجموعه ای از تکرار رشته های رشته ای است که در آن هر عضو با یک امتیاز همراه است. موارد با توجه به نمره آنها طبقه بندی می شوند.

پرونده views.py برنامه را ویرایش کنید و نمای image_detail را به شرح زیر بنویسید:

```
def image_detail(request, id, slug):
    image = get_object_or_404(Image, id=id, slug=slug)
    # increment total image views by 1
    total_views = r.incr('image:{}:views'.format(image.id))
    # increment image ranking by 1
    r.zincrby('image_ranking', image.id, 1)
    return render(request,
                  'images/image/detail.html',
                  {'section': 'images',
                   'image': image,
                   'total_views': total_views})
```

ما از دستور `zincrby()` برای ذخیره نمای تصویر در یک مجموعه مرتب شده با تصویر استفاده می‌کنیم: کلید رتبه بندی. ما شناسه تصویر و نمره مربوط به ۱ را ذخیره می‌کنیم که به مجموع امتیاز این عنصر در مجموعه مرتب شده اضافه می‌شود. این به ما امکان می‌دهد تا تمام نمای تصویری را در سطح جهان پیگیری کیم و مجموعه مرتب شده‌ای داشته باشیم که بر اساس تعداد بازدیدها تنظیم شده است.

اکنون برای نمایش رتبه بندی تصاویر مشاهده شده، یک نمای جدید ایجاد کنید. کد زیر را به پرونده `views.py` برنامه تصویر اضافه کنید:

```
@login_required
def image_ranking(request):
```

```

# get image ranking dictionary
image_ranking = r.zrange('image_ranking', 0, -1,
                        desc=True)[:10]
image_ranking_ids = [int(id) for id in image_ranking]
# get most viewed images
most_viewed = list(Image.objects.filter(
    id__in=image_ranking_ids))
most_viewed.sort(key=lambda x: image_ranking_ids.index(x.id))
return render(request,
              'images/image/ranking.html',
              {'section': 'images',
               'most_viewed': most_viewed})

```

نمای image_ranking به این صورت عمل می کند:

۱. ما از دستور `zrange()` برای بدست آوردن عناصر موجود در مجموعه مرتب شده استفاده می کنیم. این دستور با توجه به کمترین و بالاترین امتیاز از یک محدوده سفارشی انتظار دارد. با استفاده از ۰ به عنوان پایین ترین و ۱ به عنوان بالاترین امتیاز ، ما به Redis می گوییم که همه عناصر موجود در مجموعه مرتب شده را برگرداند. ما همچنین بازیابی = صحیح برای بازیابی عناصر دستور داده شده توسط نزولی نمره را مشخص می کنیم. سرانجام ، ما با استفاده از [:10] نتایج را قطعه قطعه می کنیم تا ۱۰ عنصر اول با بالاترین امتیاز را بدست آوریم.

۲. ما لیستی از شناسه های برگشت یافته را ایجاد می کنیم و آن را در متغیر `image_ranking_ids` به عنوان لیستی از عدد صحیح ذخیره می کنیم. ما اشیاء تصویر را برای آن شناسه ها بازیابی می کنیم و پرس و جو را مجبور می کنیم با استفاده از `sort()` اجرا شود. اجراء اجرای QuerySet بسیار مهم است زیرا ما اکنون از روش `list()` برای `sort()` کردن در آن استفاده خواهیم کرد (در این مرحله به جای QuerySet به لیستی از اشیاء نیاز داریم).

۳ اشیاء تصویر را بر اساس شاخص ظاهری آنها در رتبه بندی تصویر مرتب می کنیم. اکنون می توانیم از لیست `objects` در الگوی خود استفاده کنیم تا ۱۰ تصویر مشاهده شده را به نمایش بگذاریم.

یک الگوی جدید استفاده از تصویر ایجاد کنید و کد زیر را به آن images/image/template rank.html در داخل اضافه کنید:

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

{% block title %}Images ranking{% endblock %}

{% block content %}
<h1>Images ranking</h1>
<ol>
    {% for image in most_viewed %}
        <li>
            <a href="{{ image.get_absolute_url }}">
                {{ image.title }}
            </a>
        </li>
    {% endfor %}
</ol>
{% endblock %}
```

قالب بسیار ساده است. ما از اشیاء تصویر موجود در لیست بیشترین بازدید تکرار می کنیم و نام آنها را از جمله پیوند به صفحه جزئیات تصویر نمایش می دهیم.

Bookmarks My dashboard Images People Hello Antonio, Logout

Images ranking

1. [Chick Corea](#)
2. [Louis Armstrong](#)
3. [Al Jarreau](#)
4. [Django Reinhardt](#)
5. [Django and Duke](#)

عالی! شما فقط با ردیس رتبه ای ایجاد کرده اید.

مراحل بعدی با Redis

جایگزینی برای پایگاه داده SQL شما نیست ، بلکه حافظه سریع است Redis

ذخیره سازی مناسب تر برای کارهای خاص است. آن را به پشته خود بیفزایید و وقتی واقعاً احساس کردید که لازم است از آن استفاده کنید. در زیر چند سناریو وجود دارد که در آن Redis بسیار مناسب است:

شمارش: همانطور که مشاهده کردید ، کنترل پیشخوان ها با Redis بسیار آسان است. برای شمارش موارد می توانید از incr () و incrby () استفاده کنید.

ذخیره آخرین موارد: با استفاده از lpush () و rpush () می توانید موارد ابتدای / انتهای لیست را اضافه کنید. اولین / آخرین عنصر را با استفاده از lpop / rpop () حذف کرده و برگردانید. می توانید طول لیست را با استفاده از ltrim () کوتاه کنید تا طول آن حفظ شود.

صف: علاوه بر دستورات فشار و پاپ ، Redis دستورات صفحه مسدود کردن را ارائه می دهد.

حافظه پنهان: استفاده از منقضی () و منقضی () به شما امکان می دهد از Redis به عنوان حافظه نهان استفاده کنید. همچنین می توانید باکتری حافظه نهان ثانویه Redis را برای Django پیدا کنید.

Pub / sub: Redis دستورهایی را برای اشتراک / لغو اشتراک و ارسال پیام به کانالها فراهم می کند.

رتبه بندی و تابلوهای راهنمای مجموعه های مرتب شده با امتیاز با ایجاد امتیاز برای ایجاد تابلوهای مدیران بسیار آسان است.

ردیابی زمان واقعی: سرعت سریع O / ۱ ردیس آن را برای سناریوهای زمان واقعی کامل می کند.

خلاصه

در این فصل ، شما یک سیستم دنبال کننده و یک جریان فعالیت کاربر ایجاد کرده اید. شما آموخته اید که چگونه سیگنال های Django کار می کنند و Redis را در پروژه خود ادغام می کنند.

در فصل بعد نحوه ساخت یک فروشگاه آنلاین را یاد خواهید گرفت. شما می توانید کاتالوگ محصول ایجاد کرده و با استفاده از جلسات ، سبد خرید ایجاد کنید. شما همچنین یاد می گیرید که چگونه کارها را با استفاده از کرفس غیر همزمان انجام دهید.

فصل هفتم

ساخت یک فروشگاه آنلاین

در فصل قبل ، شما یک سیستم پیرو ایجاد کرده اید و یک جریان فعالیت کاربر ایجاد کرده اید. همچنین آموخته اید که چگونه سیگنال های Django کار می کنند و Redis را در پروره خود ادغام می کنند تا نمایهای تصویر را بشمارید. در این فصل نحوه ساخت یک فروشگاه اینترنتی اساسی را یاد خواهید گرفت. شما می توانید کاتالوگ محصولات ایجاد کرده و سبد خرید را با استفاده از جلسات جنگو پیاده سازی کنید. شما همچنین می آموزید که چگونه پردازنده های متن سفارشی را ایجاد کنید و کارهای ناهمزمان را با استفاده از کرفس شروع کنید.

در این فصل یاد می گیرید که:

ایجاد کاتالوگ محصول

با استفاده از جلسات جنگو یک سبد خرید بسازید

سفارشات مشتری را مدیریت کنید

اعلان های ناهمزمان را به مشتریانی که از کرفس استفاده می کنند ارسال کنید

ایجاد یک پروژه فروشگاه آنلاین

ما قصد داریم با ساختن یک فروشگاه اینترنتی آنلاین با یک پروژه جدید جنگو شروع کنیم. کاربران ما قادر خواهند بود از طریق فروشگاه محصولات فهرست شده و محصولات را به سبد خرید اضافه کنند. سرانجام ، آنها می توانند از سبد خرید خارج شده و سفارش دهند. در این فصل ویژگی های زیر در مورد یک فروشگاه آنلاین آورده شده است:

ایجاد مدل های فروشگاه محصولات ، افزودن آنها به سایت مدیریت و ایجاد نمای اصلی برای نمایش کاتالوگ

ساختن یک سیستم سبد خرید با استفاده از جلسات Django به کاربران امکان می دهد تا ضمن مروز سایت ، محصولات منتخب را انتخاب کنند

ایجاد فرم و کارایی برای سفارش دادن در سایت

ارسال یک تأیید نامه ناهمزمان به کاربران هنگام سفارش

یک پوسته را باز کنید ، یک محیط مجازی برای پروژه جدید ایجاد کنید و آن را با دستورات زیر فعال کنید:

```
mkdir env  
virtualenv env/myshop  
source env/myshop/bin/activate
```

با باز کردن یک پوسته و اجرای دستورات زیر ، یک پروژه جدید به نام myshop را با یک برنامه به نام shop شروع کنید:

```
pip install Django
```

پرونده settings.py پروژه خود را ویرایش کنید و برنامه فروشگاه را به تنظیمات INSTALLED_APPS به شرح زیر اضافه کنید:

```
django-admin startproject myshop  
cd myshop/  
django-admin startapp shop
```

برنامه شما اکنون برای این پروژه فعال است. باید مدل هایی را برای کاتالوگ محصول تعریف کنیم.

```
INSTALLED_APPS = [  
    # ...  
    'shop.apps.ShopConfig',  
]
```

ایجاد مدل‌های فروشگاه محصولات

فروشگاه ما شامل محصولاتی خواهد بود که در دسته های مختلف سازماندهی می شوند. هر محصول دارای نام ، توضیحات اختیاری ، تصویر اختیاری ، قیمت و درسترس خواهد بود. پرونده model.py برنامه فروشگاه را که اخیراً ایجاد کرده اید ویرایش کنید و کد زیر را اضافه کنید:

```
from django.db import models

class Category(models.Model):
    name = models.CharField(max_length=200,
                           db_index=True)
    slug = models.SlugField(max_length=200,
                           unique=True)

    class Meta:
        ordering = ('name',)
        verbose_name = 'category'
        verbose_name_plural = 'categories'

    def __str__(self):
        return self.name

class Product(models.Model):
    category = models.ForeignKey(Category,
                                 related_name='products',
                                 on_delete=models.CASCADE)
    name = models.CharField(max_length=200, db_index=True)
    slug = models.SlugField(max_length=200, db_index=True)
    image = models.ImageField(upload_to='products/%Y/%m/%d',
                             blank=True)
    description = models.TextField(blank=True)
    price = models.DecimalField(max_digits=10, decimal_places=2)
    available = models.BooleanField(default=True)
    created = models.DateTimeField(auto_now_add=True)
    updated = models.DateTimeField(auto_now=True)
```

```
class Meta:  
    ordering = ('name',)  
    index_together = (('id', 'slug'),)  
  
    def __str__(self):  
        return self.name
```

این مدل های دسته بندی و محصول هستند. مدل Category از یک فیلد نام و یک میدان منحصر به فرد مثل حلقه ای تشکیل شده است (منحصر به فرد به معنی ایجاد یک فهرست) است. قسمت های مدل محصول به شرح زیر است: رده: به مدل طبقه. این یک رابطه چند به یک است: یک محصول به یک دسته تعلق دارد و یک گروه شامل چندین محصول است.

نام: نام محصول.

slug: شلخته این محصول برای ساختن URL های زیبا است.

تصویر: تصویر محصول اختیاری.

توضیحات: توضیحات اختیاری از محصول.

Price: در این زمینه از یک دهدی اعشاری Python استفاده می شود تا یک عدد اعشاری با دقت ثابت را ذخیره کنید. حداقل تعداد رقم ها (از جمله مکان های اعشاری) با استفاده از ویژگی max_digits و مکان های اعشاری با ویژگی اعشاری dhjet تنظیم می شود.

در دسترس: مقدار بولی که نشان می دهد محصول موجود است یا نه. از آن برای فعال یا غیرفعال کردن محصول موجود در فروشگاه استفاده می شود.

ایجاد شده: این قسمت هنگام ایجاد شیء ذخیره می شود.

به روز شده: این قسمت هنگامی که آخرین شیء به روز شد، ذخیره می کند.

برای زمینه قیمت، ما از DecimalField به جای FloatField استفاده می کنیم تا از مشکلات گرد کردن جلوگیری کنیم.

در کلاس متأ مدل Product ، از گزینه index_together استفاده می کنیم تا یک شاخص برای زمینه های شناسه و slug با هم مشخص کنیم. ما این شاخص را تعریف می کنیم زیرا قصد داریم محصولات را با استفاده از id و slug پرس و جو کنیم. هر دو زمینه با هم فهرست بندی می شوند تا عملکردهای نمایش داده شد که از این دو زمینه استفاده می کنند را بهبود بخشنند.

از آنجاکه قصد داریم در مدل های خود با تصاویر مقابله کنیم ، پوسته را باز کرده و pillowرا با دستور زیر نصب می کنید:

```
pip install Pillow=
```

اکنون ، دستور بعدی را اجرا کنید تا مهاجرت های اولیه را برای پروژه خود ایجاد کنید:

```
python manage.py makemigrations
```

خروجی زیر را مشاهده خواهید کرد:

```
Migrations for 'shop':  
  shop/migrations/0001_initial.py  
    - Create model Category  
    - Create model Product  
    - Alter index_together for product (1 constraint(s))
```

دستور بعدی را برای همگام سازی بانک اطلاعات اجرا کنید:

```
python manage.py migrate
```

خروجی را مشاهده خواهید کرد که شامل خط زیر است:

```
Applying shop.0001_initial... OK
```

اکنون پایگاه داده با مدل‌های شما همگام سازی شده است.

ثبت مدل‌های کاتالوگ در سایت سرپرست

بیایید مدل‌های خود را به سایت مدیریت اضافه کنیم تا بتوانیم به راحتی دسته‌ها و محصولات را مدیریت کنیم. فایل برنامه فروشگاه را ویرایش کنید و کد زیر را به آن اضافه کنید:

```
from django.contrib import admin
from .models import Category, Product

@admin.register(Category)
class CategoryAdmin(admin.ModelAdmin):
    list_display = ['name', 'slug']
    prepopulated_fields = {'slug': ('name',)}


@admin.register(Product)
class ProductAdmin(admin.ModelAdmin):
    list_display = ['name', 'slug', 'price',
                    'available', 'created', 'updated']
    list_filter = ['available', 'created', 'updated']
    list_editable = ['price', 'available']
    prepopulated_fields = {'slug': ('name',)}
```

به یاد داشته باشید که ما از ویژگی prepopulated_fields استفاده می‌کنیم تا زمینه‌هایی را تعیین کنیم که مقدار به طور خودکار با استفاده از مقدار فیلد‌های دیگر تنظیم شود. همانطور که قبلاً مشاهده کرده اید، این کار برای تولید اسلايدها مناسب است. ما در کلاس ProductAdmin از ویژگی list_editable استفاده می‌کنیم تا زمینه‌هایی را که می‌توان از صفحه

نمایش لیست سایت مدیریت ویرایش کرد ، تنظیم کنیم. این به شما امکان می دهد چندین ردیف را پکجا ویرایش کنید. هر فیلد در `list_display` نیز باید در ویژگی `list_display` باشد زیرا فقط فیلدهای نمایش داده شده قابل ویرایش هستند.

اکنون با استفاده از دستور زیر یک `superuser` برای سایت خود ایجاد کنید:

```
python manage.py createsuperuser
```

سرور توسعه را با دستور `python management.py runserver` شروع کنید.
وارد شوید. با استفاده از رابط مدیریت ، یک طبقه بندی جدید و محصول را اضافه کنید. صفحه لیست تغییر محصول
صفحه مدیریت ، اینگونه خواهد بود:

The screenshot shows the Django admin interface for a 'Products' model. At the top, there's a success message: 'The product "Green tea" was added successfully.' Below this, the main area displays a table with one row of data. The columns are labeled: NAME, SLUG, PRICE, STOCK, AVAILABLE, CREATED, and UPDATED. The single row shows: Green tea, green-tea, 30, 22, Yes, Dec 5, 2017, 6:17 p.m., Dec 5, 2017, 6:17 p.m.. To the right of the table is a sidebar titled 'FILTER' containing three sections: 'By available' (with 'All', 'Yes', and 'No' options), 'By created' (with 'Any date', 'Today', 'Past 7 days', 'This month', and 'This year' options), and 'By updated'.

Building catalog views

برای نمایش کاتالوگ محصول ، باید یک لیست ایجاد کنیم تا لیست تمام محصولات یا فیلتر محصولات بر اساس یک دسته خاص باشد. پرونده `views.py` برنامه فروشگاه را ویرایش کنید و کد زیر را به آن اضافه کنید:

```
from django.shortcuts import render, get_object_or_404
from .models import Category, Product

def product_list(request, category_slug=None):
    category = None
    categories = Category.objects.all()
    products = Product.objects.filter(available=True)
    if category_slug:
        category = get_object_or_404(Category, slug=category_slug)
        products = products.filter(category=category)
    return render(request,
                  'shop/product/list.html',
                  {'category': category,
                   'categories': categories,
                   'products': products})
```

ما با استفاده از `available=True` QuerySet را با استفاده از `available=True` جرامی کنیم تا فقط محصولات موجود را بازیابی کنیم. ما از یک پارامتر اختیاری برای فیلتر کردن محصولات به صورت دسته بندی خاص استفاده می کنیم. `category_slug` برای بازیابی و نمایش یک محصول واحد نیز به نمایش نیاز داریم. نمای زیر را به پرونده `views.py` اضافه کنید:

```
def product_detail(request, id, slug):
    product = get_object_or_404(Product,
                                id=id,
                                slug=slug,
                                available=True)
    return render(request,
                  'shop/product/detail.html',
                  {'product': product})
```

نمای Product_detail برای بازیابی نمونه از پارامترهای id و slug انتظار دارد. ما می توانیم این نمونه را فقط از طریق شناسه دریافت کنیم زیرا این یک ویژگی منحصر به فرد است. با این وجود ، ما برای ایجاد URL های مناسب برای سئو برای محصولات ، URL را در URL وارد می کنیم.

پس از ساختن لیست محصولات و نمایش جزئیات ، باید الگوهای URL را برای آنها تعریف کنیم. یک پرونده جدید را درون فهرست برنامه کاربردی فروشگاه ایجاد کنید و نام آن را urls.py نام بخواهید. کد زیر را به آن اضافه کنید:

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

app_name = 'shop'

urlpatterns = [
    path('', views.product_list, name='product_list'),
    path('<slug:category_slug>/', views.product_list,
         name='product_list_by_category'),
    path('<int:id>/<slug:slug>/', views.product_detail,
         name='product_detail'),
]
```

اینها الگوهای URL برای فروشگاه محصولات ما هستند. ما دو الگوی URL مختلف برای نمایه Product_list تعریف کرده ایم: الگویی به نام product_list ، که نمای Product_list را بدون هیچ پارامتر فراخوانی می کند. والگویی به نام category_slug ، که یک پارامتر را برای تصفیه محصولات با توجه به یک دسته معین ، نمای را فراهم می کند. ما برای بازیابی یک محصول خاص ، الگویی برای نمای product_detail اضافه کردیم که پارامترهای id و slug را به نمای منتقل می کند.

پرونده urls.py پروژه myshop را ویرایش کنید تا به شکل زیر ظاهر شود:

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

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

در الگوهای اصلی URL پروژه ، ما URL هایی را برای برنامه فروشگاه تحت یک نام تجاری سفارشی به نام shop درج خواهیم کرد.

اکنون پرونده model.py برنامه فروشگاه را ویرایش کنید ، عملکرد() را وارد کنید و یک روش get_absolute_url را به مدل های دسته بندی و محصول به شرح زیر اضافه کنید:

```
from django.urls import reverse
# ...
class Category(models.Model):
    # ...
    def get_absolute_url(self):
        return reverse('shop:product_list_by_category',
                      args=[self.slug])

class Product(models.Model):
    # ...
    def get_absolute_url(self):
        return reverse('shop:product_detail',
                      args=[self.id, self.slug])
```

همانطور که از قبل می دانید ، get_absolute_url کنوانسیونی برای بازیابی URL برای یک شیء معین است. در اینجا ، ما از الگوهای URL هایی که تازه در پرونده urls.py تعریف کرده ایم استفاده خواهیم کرد.

ایجاد الگوهای کاتالوگ

حال ، ما نیاز به ایجاد قالب هایی برای لیست محصولات و نمایش جزئیات داریم. دایرکتوری و ساختار پرونده زیر را در فهرست راهنمای برنامه فروشگاه ایجاد کنید:

```
templates/
shop/
base.html
product/
list.html
detail.html
```

```
templates/
shop/
base.html
product/
list.html
detail.html
```

ما باید یک الگوی پایه را تعریف کنیم و سپس آن را در لیست محصولات و الگوهای جزئیات گسترش دهیم. الگوی /shop/base.html را ویرایش کنید و کد زیر را به آن اضافه کنید:

```
{% load static %}  
<!DOCTYPE html>  
<html>  
<head>  
    <meta charset="utf-8" />  
    <title>{% block title %}My shop{% endblock %}</title>  
    <link href="{% static "css/base.css" %}" rel="stylesheet">  
</head>  
<body>  
    <div id="header">  
        <a href="/" class="logo">My shop</a>  
    </div>  
    <div id="subheader">  
        <div class="cart">  
            Your cart is empty.  
        </div>  
    </div>  
    <div id="content">  
        {% block content %}  
        {% endblock %}  
    </div>  
</body>
```

این الگوی پایه ای است که ما برای فروشگاه خود استفاده خواهیم کرد. به منظور گنجاندن سبک های CSS و تصاویری که توسط قالب ها استفاده می شود ، باید فایل های استاتیک همراه با این فصل را که در استاتیک / فهرست راهنمای برنامه فروشگاه قرار دارد ، کپی کنید. آنها را در همان مکان در پروژه خود کپی کنید.

الگوی shop/product/list.html را ویرایش کنید و کد زیر را به آن اضافه کنید:

```

{% extends "shop/base.html" %}
{% load static %}

{% block title %}
    {% if category %}{{ category.name }}{% else %}Products{% endif %}
{% endblock %}

{% block content %}
<div id="sidebar">
    <h3>Categories</h3>
    <ul>
        <li {% if not category %}class="selected"{% endif %}>
            <a href="{% url "shop:product_list" %}">All</a>
        </li>
        {% for c in categories %}
            <li {% if category.slug == c.slug %}class="selected"
            {% endif %}>
                <a href="{{ c.get_absolute_url }}">{{ c.name }}</a>
            </li>
        {% endfor %}
    </ul>
</div>
<div id="main" class="product-list">
    <h1>{% if category %}{{ category.name }}{% else %}Products
    {% endif %}</h1>
    {% for product in products %}
        <div class="item">
            <a href="{{ product.get_absolute_url }}">
                
            </a>
            <a href="{{ product.get_absolute_url }}">{{ product.name }}</a>
            <br>
            ${{ product.price }}
        </div>
    {% endfor %}
</div>
{% endblock %}

```

این الگوی لیست محصولات است. این الگوی فروشگاه shop / base.html را گسترش می دهد و از متغیر متن متغیرها استفاده می کند تا همه دسته ها را در یک نوار کناری و محصولات برای نمایش محصولات صفحه فعلی نمایش دهد.

از الگوی مشابه برای هر دو استفاده می شود: لیست تمام محصولات موجود و لیست محصولات فیلتر شده توسط یک دسته. از آنجا که قسمت تصویر مدل Product می تواند خالی باشد ، لازم است برای محصولاتی که تصویری ندارند ،

یک تصویر پیش فرض تهیه کنیم. تصویر در فهرست پروندهای استاتیک ما با مسیر نسبی img / no_image.png قرار دارد.

از آنجاکه ما برای ذخیره سازی تصاویر محصول از ImageField استفاده می کنیم ، به سرور توسعه نیاز داریم تا فایلهای تصویری آپلود شده را ارائه دهد.

پرونده تنظیمات myshop را ویرایش کنید و تنظیمات زیر را اضافه کنید:

```
MEDIA_URL = '/media/'  
MEDIA_ROOT = os.path.join(BASE_DIR, 'media/')
```

MEDIA URL URL اصلی است که فایل‌های رسانه‌ای بارگذاری شده توسط کاربران را ارائه می‌دهد.

MEDIA_ROOT مسیری محلی است که این پرونده ها در آن قرار دارند ، که ما با آماده سازی پویا متغیر BASE_DIR را ساختیم.

برای اینکه Django بتواند فایلهای رسانه آپلود شده را با استفاده از سرور توسعه دهنده ارائه دهد ، پرونده اصلی urls.py برای myshop را ویرایش کنید و کد زیر را به آن اضافه کنید:

به پاد داشته باشید که ما در حین توسعه فقط به پرونده های استاتیک خدمت می کنیم. در یک محیط تولید ، شما هرگز نباید فایلهای استاتیک را با Django سرو کنید.

با استفاده از سایت مدیریت ، دو محصول را به فروشگاه خود اضافه کنید و در مرورگر خود `http://127.0.0.1:8000` را باز کنید. صفحه لیست محصولات را مشاهده خواهید کرد که به شکل زیر است:

My shop

Your cart is empty.

Products

Categories

All

Tea

Product	Price
Green tea	\$30
Red tea	\$45.5
Tea powder	\$21.2

اگر محصولی را با استفاده از سایت مدیریت ایجاد می کنید و تصویری برای آن بارگذاری نمی کنید ، به جای آن تصویر پیش فرض no_image.png نمایش داده می شود:

NO IMAGE
AVAILABLE



Green tea

\$30

Red tea

\$45.5

Tea powder

\$21.2

باید الگوی جزئیات محصول را ویرایش کنیم. [shop/product/detail.html](#)

قالب را اضافه کنید و کد زیر را به آن اضافه کنید:

```
{% extends "shop/base.html" %}  
{% load static %}  
  
{% block title %}  
    {{ product.name }}  
{% endblock %}  
  
{% block content %}  
    <div class="product-detail">  
          
        <h1>{{ product.name }}</h1>  
        <h2><a href="{{ product.category.get_absolute_url }}>{{ product.category }}</a></h2>  
        <p class="price">${{ product.price }}</p>  
        {{ product.description|linebreaks }}  
    </div>  
{% endblock %}
```

ما با استفاده از متده استفاده از متد `get_absolute_url` در موضوع طبقه بندی مرتبط برای نمایش محصولات موجود متعلق به همان دسته هستیم.

اکنون در مرورگر خود `http://127.0.0.1:8000/` را باز کرده و روی هر محصول کلیک کنید تا صفحه جزئیات محصول را ببینید. به شرح زیر خواهد بود:

My shop

Your cart is empty.



Red tea

Tea

\$45.5

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. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

اکنون شما یک فروشگاه محصولات اصلی ایجاد کرده اید.

ساخت سبد خرید

بعد از ساختن کاتالوگ محصول ، مرحله بعدی ایجاد سبد خرید است تا کاربران بتوانند کالاهای را که می خواهند خریداری کنند انتخاب کنند. سبد خرید به شما امکان می دهد محصولات را انتخاب کرده و مبالغه مورد نظر خود را سفارش دهید و سپس این اطلاعات را به طور موقت ذخیره کنید ، در حالی که آنها سایت را مرور می کنند تا اینکه در نهایت سفارش خود را انجام دهند. سبد خرید باید در این جلسه ادامه داشته باشد تا موارد ویژیت در طول بازدید کاربر حفظ شود.

ما از چارچوب جلسه Django برای ادامه کار در سبد استفاده خواهیم کرد. سبد خرید در جلسه تا زمان اتمام نگه داشته می شود یا کاربر از سبد چک خارج می شود. ما همچنین نیاز به ساخت مدل های اضافی جنگو برای سبد خرید و وسائل آن داریم.

استفاده از جلسات جنگو

Django چارچوبی برای جلسه ارائه می دهد که جلسات ناشناس و کاربر را پشتیبانی می کند. چارچوب جلسه به شما امکان می دهد داده های دلخواه را برای هر بازدید کننده ذخیره کنید. داده های جلسه در سمت سرور ذخیره می شوند و کوکی ها حاوی شناسه جلسه هستند مگر اینکه از موتور جلسه مبتنی بر کوکی استفاده کنید. میان افزار جلسه ارسال و دریافت کوکی ها را مدیریت می کند. موتور جلسه پیش فرض داده های جلسه را در دیتابیس ذخیره می کند ، اما می توانید بین موتورهای جلسه مختلف انتخاب کنید.

برای استفاده از جلسات ، باید اطمینان حاصل کنید که تنظیمات MIDDLEWARE پروژه شما شامل "django.contrib.sessions.middleware.SessionMiddleware" است.

این میان افزار مدیریت جلسات را بر عهده دارد. هنگام ایجاد یک پروژه جدید با استفاده از دستور شروع پروژه ، به طور پیش فرض به تنظیمات MIDDLEWARE اضافه می شود.

میان افزار جلسه باعث می شود جلسه فعلی در موضوع درخواست موجود باشد. شما می توانید با استفاده از request.session به جلسه فعلی دسترسی پیدا کنید ، آن را مانند یک فرهنگ لغت پایتون برای ذخیره و بازیابی اطلاعات جلسه ، به جلسه فعلی وارد کنید. فرهنگ لغت جلسه هر شیء پایتون را بصورت پیش فرض که می تواند به JSON سریال شود ، می پذیرد. می توانید متغیر را در جلسه مانند این تنظیم کنید:

```
request.session['foo'] = 'bar'
```

یک کلید جلسه را به شرح زیر بازیابی کنید:

```
request.session.get('foo')
```

کلیدی را که قبلاً در جلسه ذخیره کرده اید به شرح زیر حذف کنید:

```
del request.session['foo']
```

شما فقط می توانید با `request.session` مانند یک فرهنگ لغت استاندارد پایتون رفتار کنید.

تنظیمات جلسه

چندین تنظیمات وجود دارد که می توانید برای پیکربندی جلسات پروژه خود استفاده کنید. مهمترین آنها `SESSION_ENGINE` است. این تنظیم به شما امکان می دهد مکانی را که در آن جلسات ذخیره می شود تنظیم کنید. به طور پیش فرض ، Django جلسات را در پایگاه داده با استفاده از مدل `django.contrib.sessions.Session` برنامه ذخیره می کند.

Django گزینه های زیر را برای ذخیره داده های جلسه ارائه می دهد:

جلسات بانک اطلاعاتی: داده های جلسه در پایگاه داده ذخیره می شود.

این موتور جلسه پیش فرض است.

جلسات مبتنی بر پرونده: داده های جلسه در سیستم فایل ذخیره می شود.

جلسات ذخیره شده: داده های جلسه در یک پس زمینه حافظه نهان ذخیره می شود. با استفاده از تنظیمات `CACHES` می توانید باطن حافظه نهان را تعیین کنید. ذخیره داده های جلسه در یک سیستم حافظه پنهان بهترین عملکرد را دارد.

جلسات ذخیره شده پایگاه داده: داده های جلسه در حافظه نهان و پایگاه داده ذخیره می شوند. در صورت خواندن داده ها در حافظه نهان ، از پایگاه داده فقط استفاده می کند.

جلسات مبتنی بر کوکی: داده های جلسه در کوکی هایی که به مرورگر ارسال می شوند ذخیره می شوند.

می توانید جلسات را با تنظیمات خاص سفارشی کنید. در اینجا برخی از تنظیمات مهم مربوط به جلسه آورده شده است:

SESSION_COOKIE_AGE: مدت زمان کوکی های جلسه در ثانیه.

مقدار پیش فرض 1209600 (دو هفته) است.

SESSION_COOKIE_DOMAIN: دامنه مورد استفاده برای کوکی های جلسه. برای فعال کردن کوکی های دامنه متقابل با استفاده از None برای یک کوکی دامنه استاندارد ، این را روی mydomain.com تنظیم کنید.

SESSION_COOKIE_SECURE: یک بولی که نشان می دهد کوکی فقط در صورت اتصال اتصال HTTPS ارسال می شود.

SESSION_EXPIRE_AT_BROWSER_CLOSE: یک بول که نشان می دهد جلسه باید بسته شود مروگر بسته می شود.

SESSION_SAVE_EVERY_REQUEST: یک بولی که اگر صحیح باشد ، جلسه را در هر درخواست در پایگاه داده ذخیره می کند. منقضی جلسه نیز هر بار ذخیره می شود.

می توانید تمام تنظیمات جلسه و مقادیر پیش فرض آنها را در <https://docs.djangoproject.com/fa/2.0/ref/settings/#sessions> مشاهده کنید.

منقضی شدن جلسه

با استفاده از تنظیمات SESSION_EXPIRE_AT_BROWSER_CLOSE می توانید جلسات مروگر یا جلسات مدام را انتخاب کنید. به طور پیش فرض روی False تنظیم شده است و مدت زمان جلسه را به مقدار ذخیره شده در تنظیمات SESSION_EXPIRE_AT_BROWSER_CLOSE مجبور می کند. اگر True را روی SESSION_COOKIE_AGE تنظیم کنید ، با بسته شدن کاربر مروگر ، جلسه منقضی می شود و تنظیم SESSION_COOKIE_AGE هیچ تاثیری نخواهد داشت.

برای نوشتتن مدت زمان جلسه فعلی می توانید از روش session.set_expiry() درخواست استفاده کنید.

ذخیره سبد خرید در جلسات

ما باید یک ساختار ساده ایجاد کنیم که بتواند برای JSON سریال سازی شود برای ذخیره آیتم های سبد خرید در یک جلسه. سبد خرید باید داده های زیر را برای هر مورد موجود در آن داشته باشد:

شناسه نمونه محصول

مقدار انتخاب شده برای محصول

قیمت واحد محصول

از آنجاکه ممکن است قیمت محصول متفاوت باشد ، ما می توانیم هنگام اضافه کردن به سبد خرید ، قیمت محصول را به همراه خود محصول حفظ کنیم. با این کار ، وقتی کاربران آن را به سبد خرید خود اضافه کنند ، از قیمت فعلی محصول استفاده می کنیم ، مهم نیست که قیمت محصول پس از آن تغییر کند.

اکنون شما باید برای ایجاد سبد خرید و ایجاد ارتباط با آنها سبد خرید باید به صورت زیر کار کنید:

وقتی سبد مورد نیاز است ، بررسی می کنیم که آیا یک کلید جلسه سفارشی تنظیم شده است یا خیر. اگر هیچ سبد در جلسه تنظیم نشده باشد ، یک سبد خرید جدید ایجاد می کنیم و آن را در کلید جلسه سبد ذخیره می کنیم. برای درخواست های پی در پی ، همان چک را انجام می دهیم و وسائل سبد خرید را از کلید جلسه سبد خرید دریافت می کنیم. ما موارد سبد خرید را از جلسه و اشیاء مربوط به محصول آنها از پایگاه داده بازیابی می کنیم.

پرونده settings.py پروژه خود را ویرایش کنید و تنظیمات زیر را به آن اضافه کنید:

```
CART_SESSION_ID = 'cart'
```

این کلیدی است که ما می خواهیم از آن برای ذخیره سبد خرید در جلسه کاربر استفاده کنیم. از آنجاکه جلسات جنگو به ازای هر بازدید کننده اداره می شود ، می توانیم از کلیدی همان جلسه سبد خرید برای همه جلسات استفاده کنیم.

بایاید یک برنامه برای مدیریت چرخ دستی های خرید ایجاد کنیم. ترمینال را باز کنید و یک برنامه جدید ایجاد کنید ، دستور زیر را از فهرست پروژه اجرا کنید:

```
python manage.py startapp cart
```

سپس پرونده setting.py پروژه خود را ویرایش کرده و برنامه جدید را به تنظیمات INSTALLED_APPS اضافه کنید:

```
INSTALLED_APPS = [
    # ...
    'shop.apps.ShopConfig',
    'cart.apps.CartConfig',
]
```

یک پرونده جدید را درون فهرست برنامه کاربردی سبد ایجاد کرده و نام آن را cart.py بگذارید. کد زیر را به آن اضافه کنید:

```
from decimal import Decimal
from django.conf import settings
from shop.models import Product

class Cart(object):

    def __init__(self, request):
        """
        Initialize the cart.
        """
        self.session = request.session
        cart = self.session.get(settings.CART_SESSION_ID)
        if not cart:
            cart = self.session[settings.CART_SESSION_ID] = {}
        self.cart = cart

    def add(self, product, quantity=1, update_quantity=False):
        product_id = str(product.id)
        if product_id not in self.cart:
            self.cart[product_id] = {'quantity': 0, 'price': str(product.price)}
        if update_quantity:
            self.cart[product_id]['quantity'] = quantity
        else:
            self.cart[product_id]['quantity'] += quantity
        self.save()

    def save(self):
        self.session.modified = True

    def remove(self, product):
        product_id = str(product.id)
        if product_id in self.cart:
            del self.cart[product_id]
            self.save()

    def get_total_price(self):
        return sum(item['price'] * item['quantity'] for item in self.cart.values())

    def get_items(self):
        from shop.models import Product
        product_ids = self.cart.keys()
        products = Product.objects.filter(id__in=product_ids)
        return products

    def clear(self):
        self.cart.clear()
        self.save()
```

```
if not cart:  
    # save an empty cart in the session  
    cart = self.session[settings.CART_SESSION_ID] = {}  
self.cart = cart
```

این کلاس Cart است که به ما امکان می دهد سبد خرید را مدیریت کنیم.

ما نیاز داریم که سبد خرید با یک موضوع درخواست اولیه شود. جلسه فعلی را با استفاده از `self.session` می کنیم تا در دسترس سایر روش‌های کلاس Cart قرار گیرد. ابتدا سعی می کنیم با استفاده از `request.session` (تنظیمات `CART_SESSION_ID`) از جلسه فعلی سبد خرید را بگیریم. اگر هیچ سبد در جلسه حضور ندارد، با تنظیم یک فرهنگ لغت خالی در جلسه، یک سبد خالی ایجاد می کنیم. ما انتظار داریم که فرهنگ لغت سبد خرید ما از شناسه محصول به عنوان کلید و فرهنگ لغت با کمیت و قیمت به عنوان مقدار برای هر کلید استفاده کند. با این کار می توانیم تضمین کنیم که یک محصول بیش از یک بار در سبد خرید اضافه نشده باشد. به این ترتیب ما همچنین روش بازیابی کالاهای سبد را ساده می کنیم.

باید روشی را برای اضافه کردن محصولات به سبد خرید یا به روزرسانی مقدار آنها ایجاد کنیم. روش‌های `(add()` و `(remove())` را به کلاس Cart اضافه کنید:

```

class Cart(object):
    # ...
    def add(self, product, quantity=1, update_quantity=False):
        """
        Add a product to the cart or update its quantity.
        """
        product_id = str(product.id)
        if product_id not in self.cart:
            self.cart[product_id] = {'quantity': 0,
                                   'price': str(product.price)}
        if update_quantity:
            self.cart[product_id]['quantity'] = quantity
        else:
            self.cart[product_id]['quantity'] += quantity
        self.save()

    def save(self):
        # mark the session as "modified" to make sure it gets saved
        self.session.modified = True

```

متدهای زیر را به عنوان ورودی در نظر می‌گیرد:

محصول: نمونه محصول برای افزودن یا به روزرسانی در سبد خرید.

کمیت: یک عدد صحیح اختیاری با مقدار محصول. این پیش فرض به ۱ است.

باشد update_quantity: این یک بولی است که نشان می‌دهد آیا باید مقدار با مقدار داده شده به روز شود (درست) یا اینکه باید مقدار جدیدی به مقدار موجود اضافه شود (غلط).

ما از شناسه محصول به عنوان کلید در فرهنگ لغت محتوای سبد استفاده می‌کنیم. ما شناسه محصول را به صورت رشته تبدیل می‌کنیم زیرا JSON از Django برای سریال‌سازی داده‌های جلسه استفاده می‌کند، و JSON فقط به نام‌های کلیدی رشته اجازه می‌دهد. شناسه محصول کلیدی و ارزشی است که ما به آن ادامه می‌دهیم یک فرهنگ لغت با تعداد و قیمت های محصول است. به منظور تهیه سریال آن، قیمت محصول از اعشار به یک رشته تبدیل می‌شود.

در آخر، ما برای صرفه جویی در سبد خرید در جلسه، از روش save() استفاده می‌کنیم.

روش ذخیره () جلسه را با استفاده از session.modified = True اصلاح می‌کند. این به جنگویی گوید که جلسه تغییر کرده است و باید ذخیره شود.

ما همچنین به روشی برای خارج کردن محصولات از سبد خرید نیاز داریم. روش زیر را به کلاس Cart اضافه کنید:

```
class Cart(object):
    # ...
    def remove(self, product):
        """
        Remove a product from the cart.
        """
        product_id = str(product.id)
        if product_id in self.cart:
            del self.cart[product_id]
            self.save()
```

روش remove() محصولی را از فرهنگ لغت سبد خرید خارج می کند و برای ذخیره به روز در سبد روش ، روش ذخیره () را فراخوانی می کند.

ما باید از طریق موارد موجود در سبد خرید آن را تکرار کنیم و به موارد مربوط به محصولات دسترسی پیدا کنیم. برای این کار می توانید یک روش __iter__() را در کلاس خود تعریف کنید. روش زیر را به کلاس Cart اضافه کنید:

```

class Cart(object):
    # ...
    def __iter__(self):
        """
        Iterate over the items in the cart and get the products
        from the database.
        """
        product_ids = self.cart.keys()
        # get the product objects and add them to the cart
        products = Product.objects.filter(id__in=product_ids)

        cart = self.cart.copy()
        for product in products:
            cart[str(product.id)]['product'] = product

        for item in cart.values():
            item['price'] = Decimal(item['price'])
            item['total_price'] = item['price'] * item['quantity']
            yield item

```

در روش `__iter__()` نمونه های `Product` را که در سبد خرید موجود است ، بازیابی می کنیم تا آنها را در موارد سبد قرار دهیم. سبد خرید فعلی را در متغیر سبد خرید کپی می کنیم و نمونه های `Product` را به آن اضافه می کنیم.

سرانجام ، ما بیش از کالاهای سبد خرید تکرار می کنیم ، قیمت کالای موردنظر را به دهدۀ تبدیل می کنیم و یک ویژگی `total_price` را به هر مورد اضافه می کنیم. اکنون ، می توانیم به راحتی از موارد موجود در سبد خرید آن را تکرار کنیم.

ما همچنین به راه برای بازگشت تعداد کل موارد موجود در سبد خرید نیاز داریم.

هنگامی که عملکرد `len()` روی یک شی اجرا می شود ، پایتون برای بازیابی طول آن ، از روش `__len__()` خود استفاده می کند. ما می خواهیم برای بازگشت تعداد کل کالاهای ذخیره شده در سبد خرید ، یک روش `__len__()` سفارشی تعریف کنیم. روش `__len__()` زیر را به آن اضافه کنید

cart class:

```
class Cart(object):
    # ...
    def __len__(self):
        """
        Count all items in the cart.
        """
        return sum(item['quantity'] for item in self.cart.values())
```

ما مقدار کمی از تمام موارد سبد را برمی گردانیم.

برای محاسبه کل هزینه کالاهای موجود در سبد خرید روش زیر را اضافه کنید:

```
class Cart(object):
    # ...
    def get_total_price(self):
        return sum(Decimal(item['price']) * item['quantity'] for item in
self.cart.values())
```

و در آخر ، روشی را برای تمیز کردن جلسه سبد خرید اضافه کنید:

```
class Cart(object):
    # ...
    def clear(self):
        # remove cart from session
        del self.session[settings.CART_SESSION_ID]
        self.save()
```

کلاس Cart ما اکنون آماده مدیریت چرخ دستی های خرید است.

ایجاد نماهای سبد خرید

اکنون که یک کلاس سبد خرید برای مدیریت سبد خرید داریم ، باید نمایی را برای اضافه کردن ، به روزرسانی یا حذف موارد از آن ایجاد کنیم. ما باید نظرات زیر را ایجاد کنیم:

نمایی برای افزودن یا به روزرسانی موارد در سبد خرید ، که می تواند مقادیر فعلی و جدید را کنترل کند نمای جهت حذف موارد از سبد خرید نمایش کالاهای سبد خرید و جمع بندی

افزودن موارد به سبد خرید

برای اضافه کردن موارد به سبد خرید ، به یک فرم نیاز داریم که به کاربر امکان می دهد کمیت را انتخاب کند. یک پرونده را درون فهرست برنامه کاربردی سبد ایجاد کرده و کد زیر را به آن اضافه کنید:

```
from django import forms

PRODUCT_QUANTITY_CHOICES = [(i, str(i)) for i in range(1, 21)]

class CartAddProductForm(forms.Form):
    quantity = forms.TypedChoiceField(
        choices=PRODUCT_QUANTITY_CHOICES,
        coerce=int)
    update = forms.BooleanField(required=False,
                                initial=False,
                                widget=forms.HiddenInput)
```

ما از این فرم برای اضافه کردن محصولات به سبد خرید استفاده خواهیم کرد. کلاس CartAddProductForm ما شامل دو بخش زیر است:

مقدار: این به کاربر اجازه می دهد مقدار بین 1- 20 را انتخاب کند. ما از یک فیلد coercer = int typedChoiceField با

برای تبدیل ورودی به یک عدد صحیح استفاده می کنیم.

به روز رسانی: با این کار می توانید مشخص کنید که آیا مقدار این مقدار به سبد موجود برای سبد کالای موجود برای این محصول (غلط) اضافه شده است یا اینکه مقدار موجود با مقدار معین باید به روز شود (درست). ما از یک ویجت

HiddenInput برای این قسمت استفاده می کنیم زیرا نمی خواهیم آن را به کاربر نمایش دهیم.

بیایید یک نمای برای اضافه کردن موارد به سبد خرید ایجاد کنیم. پرونده views.py برنامه سبد خرید را ویرایش کنید و کد زیر را به آن اضافه کنید:

```
from django.shortcuts import render, redirect, get_object_or_404
from django.views.decorators.http import require_POST
from shop.models import Product
from .cart import Cart
from .forms import CartAddProductForm

@require_POST
def cart_add(request, product_id):
    cart = Cart(request)
    product = get_object_or_404(Product, id=product_id)
    form = CartAddProductForm(request.POST)
    if form.is_valid():
        cd = form.cleaned_data
        cart.add(product=product,
                  quantity=cd['quantity'],
                  update_quantity=cd['update'])
    return redirect('cart:cart_detail')
```

این نمای افزودن محصولات به سبد خرید یا به روزرسانی مقادیر محصولات موجود است. ما از دکوراتور requ_POST استفاده می کنیم تا فقط درخواست های POST را مجاز کنیم ، زیرا این نمایش داده ها را تغییر می دهد. نمایش شناسه محصول را به عنوان پارامتر دریافت می کند. ما نمونه محصول را با شناسه داده شده بازیابی می کنیم و CartAddProductForm را تأیید می کنیم. اگر فرم معتبر است ، ما محصول را در سبد خرید اضافه می کنیم یا به روز می کنیم.

جهت نمایش به URL سبد خرید هدایت می شود که محتوای سبد خرید را نشان می دهد. ما قصد داریم به زودی نمای cart_detail را ایجاد کنیم.

ما همچنین برای مشاهده موارد از سبد خرید به یک نمایش نیاز داریم. کد زیر را به پرونده views.py برنامه سبد خرید اضافه کنید:

```
def cart_remove(request, product_id):
    cart = Cart(request)
    product = get_object_or_404(Product, id=product_id)
    cart.remove(product)
    return redirect('cart:cart_detail')
```

نماد cart_remove شناسه محصول را به عنوان پارامتر دریافت می کند. نمونه محصول را با شناسه داده شده بازیابی می کنیم و محصول را از سبد خرید خارج می کنیم. سپس ، کاربر را به URL cart_detail هدایت می کنیم.

در آخر ، برای نمایش سبد خرید و موارد آن نیاز به نمایش داریم. نمای زیر را به پرونده views.py برنامه سبد خرید اضافه کنید:

```
def cart_detail(request):
    cart = Cart(request)
    return render(request, 'cart/detail.html', {'cart': cart})
```

نماد cart_detail برای نمایش آن به سبد خرید فعلی می رسد.

ما برای افزودن موارد به سبد خرید ، به روزرسانی مقادیر ، حذف موارد از سبد خرید و نمایش محتوای سبد نمایش ، ایجاد کرده ایم. بباید الگوهای URL را برای این نمایش ها اضافه کنیم. یک پرونده جدید را درون فهرست برنامه کاربردی سبد ایجاد کرده و نام آن را urls.py بنام کنید. آدرس های زیر را به آن اضافه کنید:

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

app_name = 'cart'

urlpatterns = [
    path('', views.cart_detail, name='cart_detail'),
    path('add/<int:product_id>/',
         views.cart_add,
         name='cart_add'),
    path('remove/<int:product_id>/',
         views.cart_remove,
         name='cart_remove'),
]
```

پرونده urls.py اصلی بروزه myshop را ویرایش کنید و الگوی URL زیر را نیز اضافه کنید تا آدرس های اینترنتی سبد را درج کنید:

```
urlpatterns = [
    path('admin/', admin.site.urls),
    path('cart/', include('cart.urls', namespace='cart')),
    path('', include('shop.urls', namespace='shop')),
]
```

اطمینان حاصل کنید که این الگوی URL را قبل از الگوی shop.urls درج کرده اید ، زیرا این محدود کننده تراز حالت دوم است.

ساختن یک الگوی برای نشان دادن سبد خرید

نماهای و cart_remove هیچ الگویی ارائه نمی دهند ، اما برای نمایش موارد و وسایل سبد خرید باید یک الگوی برای نمایش cart_detail ایجاد کنیم.

ساختر پرونده زیر را در فهرست برنامه کاربردی سبد ایجاد کنید:

```
templates/  
    cart/  
        detail.html
```

الگوی cart/detail.html را ویرایش کنید و کد زیر را به آن اضافه کنید:

```
{% extends "shop/base.html" %}  
{% load static %}  
  
{% block title %}  
Your shopping cart  
{% endblock %}  
  
{% block content %}  
<h1>Your shopping cart</h1>  
<table class="cart">  
    <thead>  
        <tr>  
            <th>Image</th>  
            <th>Product</th>  
            <th>Quantity</th>  
            <th>Remove</th>  
            <th>Unit price</th>  
            <th>Price</th>  
        </tr>  
    </thead>  
    <tbody>
```

```

    {% for item in cart %}
        {% with product=item.product %}
        <tr>
            <td>
                <a href="{{ product.get_absolute_url }}>
                    
                </a>
            </td>
            <td>{{ product.name }}</td>
            <td>{{ item.quantity }}</td>
            <td><a href="{% url "cart:cart_remove" product.id %}">Remove</a></td>
            <td class="num">${{ item.price }}</td>
            <td class="num">${{ item.total_price }}</td>
        </tr>
    {% endwith %}
    {% endfor %}
    <tr class="total">
        <td>Total</td>
        <td colspan="4"></td>
        <td class="num">${{ cart.get_total_price }}</td>
    </tr>
</tbody>
</table>
<p class="text-right">
    <a href="{% url "shop:product_list" %}" class="button
    light">Continue shopping</a>
    <a href="#" class="button">Checkout</a>
</p>
{% endblock %}

```

این الگوی است که برای نمایش محتوای سبد استفاده می شود. این شامل یک جدول با موارد ذخیره شده در سبد خرید فعلی است. ما به کاربران این امکان را می دهیم که مقدار محصولات انتخاب شده را با استفاده از فرم که در نمای سبد ارسال شده است ، تغییر دهند. ما همچنانیn به کاربران این امکان را می دهیم که با تهیه لینک حذف برای هر یک از آنها ، مواردی را از سبد خرید حذف کنند.

افزودن محصولات به سبد خرید

حال باید یک دکمه Add to cart را به صفحه جزئیات محصول اضافه کنیم. پرونده views.py برنامه فروشگاه را ویرایش کنید و CartAddProductForm را به نمای Product_detail به شرح زیر اضافه کنید:

```
from cart.forms import CartAddProductForm

def product_detail(request, id, slug):
    product = get_object_or_404(Product, id=id,
                                slug=slug,
                                available=True)
    cart_product_form = CartAddProductForm()
    return render(request,
                  'shop/product/detail.html',
                  {'product': product,
                   'cart_product_form': cart_product_form})
```

قالب shop/product/detail.html برنامه فروشگاه را ویرایش کنید و فرم زیر را به قیمت محصول به شرح زیر اضافه کنید:

```
<p class="price">${{ product.price }}</p>
<form action="{% url "cart:cart_add" product.id %}" method="post">
    {{ cart_product_form }}
    {% csrf_token %}
    <input type="submit" value="Add to cart">
</form>
{{ product.description|linebreaks }}
```

اطمینان حاصل کنید که سرور توسعه با دستور `python management.py runserver` در حال اجرا است. اکنون در مرورگر خود `http://127.0.0.1:8000/` را باز کرده و به صفحه جزئیات محصول بروید. اکنون حاوی فرم برای انتخاب مقدار قبل از اضافه کردن محصول به سبد خرید است. صفحه مانند این است:

My shop

Your cart is empty.



Red tea

Tea

\$45.5

Quantity: 1

Add to cart

کمیت را انتخاب کنید و بر روی دکمه Add to cart کلیک کنید. فرم از طریق POST به نمایه `cart_add` ارسال می شود. نمایش محصول در جلسه محصول را شامل می شود، از جمله قیمت فعلی و مقدار انتخاب شده آن، کاربر را به صفحه جزئیات کارت هدایت می کند، که مانند تصویر زیر خواهد بود:

Your shopping cart

Image	Product	Quantity	Remove	Unit price	Price
	Red tea	2	Remove	\$45.5	\$91.0
Total					\$91.0

[Continue shopping](#)[Checkout](#)

به روزرسانی مقادیر محصول در سبد خرید

وقتی کاربران سبد خرید را مشاهده می کنند ، ممکن است بخواهند قبل از سفارش ، مقدار محصول را تغییر دهند. ما می خواهیم به کاربران اجازه دهیم مقادیر را از صفحه جزئیات سبد تغییر دهند.

پرونده views.py برنامه سبد خرید را ویرایش کنید و نمای سبد خرید را به این ترتیب تغییر دهید:

```
def cart_detail(request):
    cart = Cart(request)
    for item in cart:
        item['update_quantity_form'] = CartAddProductForm(
            initial={'quantity': item['quantity'],
                      'update': True})
    return render(request, 'cart/detail.html', {'cart': cart})
```

ما نمونه ای از CartAddProductForm را برای هر مورد موجود در سبد خرید ایجاد می کنیم تا تغییر مقادیر محصول را امکان پذیر کند. ما فرم را با مقدار مورد فعلی تنظیم می کنیم و قسمت به روز رسانی را روی True قرار می دهیم تا وقتی فرم را به نمای add_arsal کنیم ، مقدار فعلی با مقدار جدید جایگزین شود.

اکنون الگوی cart/detail.html بروگرامه سبد خرید را ویرایش کرده و خط زیر را بیابید:

```
<td>{{ item.quantity }}</td>
```

خط قبلی را با کد زیر جایگزین کنید:

```
<td>
  <form action="{% url "cart:cart_add" product.id %}" method="post">

    {{ item.update_quantity_form.quantity }}
    {{ item.update_quantity_form.update }}
    <input type="submit" value="Update">
    {% csrf_token %}
  </form>
</td>
```

/ را در مرورگر خود باز کنید. برای ویرایش کمیت برای هر کالای سبد ، فرم زیر را مشاهده خواهید کرد:

Your shopping cart

Image	Product	Quantity	Remove	Unit price	Price
	Red tea	<input type="button" value="2"/> <input type="button" value="Update"/>	<input type="button" value="Remove"/>	\$45.5	\$91.0
Total					\$91.0

[Continue shopping](#)[Checkout](#)

مقدار یک مورد را تغییر داده و بر روی دکمه Update کلیک کنید تا قابلیت های جدید را آزمایش کنید. همچنان می توانید با کلیک روی پیوند حذف ، یک مورد را از سبد خرید حذف کنید.

ایجاد یک پردازنده زمینه برای سبد خرید فعلی

شاید متوجه شده باشید که پیام سبد خرید شما خالی است ، حتی در موقعی که سبد کالاها حاوی موارد باشد ، در عنوان سایت نمایش داده نمی شود. ما باید به جای آن تعداد کل موارد را در سبد خرید و کل هزینه نمایش دهیم. از آنجاکه این باید در همه صفحات نمایش داده شود ، ما یک پردازنده زمینه ایجاد خواهیم کرد تا سبد فعلی را در متن درخواست قرار دهد ، صرف نظر از دیدگاهی که درخواست را پردازش می کند.

پردازنده های متن

پردازنده متن یکتابع پایتون است که شیء درخواست را به عنوان یک آرگومان درنظر می گیرد و فرهنگ لغت را اضافه می کند که به متن درخواست اضافه می شود. آنها هنگامی که شما نیاز دارید چیزی را در سطح جهانی برای همه قالب ها در دسترس قرار دهید ، مفید هستند.

به طور پیش فرض ، هنگامی که شما با استفاده از دستور startproject یک پروژه جدید ایجاد می کنید ، پروژه شما شامل پردازنده های متن قالب زیر است ، در گزینه context_processors در داخل setting :

django.template.context_processors.debug: این اشکال زدایی اشکال زدایی را تنظیم می کند

و متغیرهای sql_queries در متن نمایانگر لیست نمایش داده شدگان SQL در درخواست.

django.template.context_processors.messages: این متغیر درخواست را در متن تنظیم می کند.

django.contrib.auth.context_processors.auth: این متغیر کاربر را در درخواست تنظیم می کند.

django.contrib.messages.context_processors.messages: این متغیر پیام را در متن شامل کلیه پیام هایی که با استفاده از چارچوب پیام ها ایجاد شده اند ، تنظیم می کند.

Django همچنین csrf_processor را قادر می سازد تا از حملات جعلی درخواست متقابل سایت جلوگیری کند. این پردازنده متن در تنظیمات موجود نیست ، اما همیشه به دلایل امنیتی فعال شده است و نمی توان آن را خاموش کرد.

لیست تمام پردازنده های متن داخلی را می توانید در <https://docs.djangoproject.com/en/2.0/ref/templates/api/#built-in-template-context-processors> مشاهده کنید.

سبد خرید را در متن درخواست تنظیم کنید

بایاید یک پردازنده زمینه ایجاد کنیم تا سبد فعلی را در متن درخواست قرار دهیم. در هر الگوی قادر خواهیم بود به سبد خرید دسترسی پیدا کنیم.

یک پرونده جدید را درون فهرست برنامه کاربردی سبد ایجاد کرده و نام آن را با context_processors.py بنویسید. پردازنده های متن می توانند در هر کجای کد شما سکونت داشته باشند ، اما ایجاد آنها در اینجا کد شما را به خوبی سازماندهی می کند.

کد زیر را به پرونده اضافه کنید:

```
from .cart import Cart

def cart(request):
    return {'cart': Cart(request)}
```

پردازنده متن تابعی است که شیء درخواست را به عنوان پارامتر دریافت می کند و دیکشنری از اشیاء را که با استفاده از RequestContext ارائه شده برای همه قالب های موجود در دسترس خواهد بود. در پردازنده متن ، سبد را با استفاده از شی درخواست فوری می کنیم و آن را به عنوان متغیری به نام سبد خرید در دسترس قرار می دهیم.

پرونده settings.py پروژه خود را ویرایش کنید و cart.context_processors.cart را به گزینه داخل تنظیمات TEMPLATES اضافه کنید:

```
TEMPLATES = [
{
    'BACKEND': 'django.template.backends.django.DjangoTemplates',
    'DIRS': [],
    'APP_DIRS': True,
    'OPTIONS': {
        'context_processors': [
            # ...
            'cart.context_processors.cart',
        ],
    },
},
]
```

پردازنده متن سبد خرید هر بار الگویی با استفاده از RequestContext Django ارائه می شود. متغیر سبد در متن قالب های شما تنظیم می شود.

اکنون قالب اپلیکیشن فروشگاه را ویرایش کنید و خطوط زیر را پیدا کنید:

```
<div class="cart">
    Your cart is empty.
</div>
```

خط های قبلی را با کد زیر جایگزین کنید:

```
<div class="cart">
    {% with total_items=cart|length %}
        {% if cart|length > 0 %}
            Your cart:
            <a href="{% url "cart:cart_detail" %}">
                {{ total_items }} item{{ total_items|pluralize }},
                ${{{ cart.get_total_price }}}
            </a>
        {% else %}
            Your cart is empty.
        {% endif %}
    {% endwith %}
</div>
```

بارگیری مجدد سرور خود با استفاده از دستور management.py runserver. <http://127.0.0.1:8000>.
در مرورگر خود باز کنید و برخی از محصولات را به سبد خرید اضافه کنید.

در عنوان وب سایت می توانید تعداد کالاهای موجود در سبد خرید و کل هزینه را به شرح زیر مشاهده کنید:

My shop

Your cart: 2 items, \$91.0

ثبت سفارشات مشتری

هنگامی که یک سبد خرید بررسی شد ، باید یک سفارش را در بانک اطلاعات ذخیره کنید. سفارشات حاوی اطلاعاتی در مورد مشتریان و محصولاتی است که می خرید.

با استفاده از دستور زیر یک برنامه جدید برای مدیریت سفارشات مشتری ایجاد کنید:

```
python manage.py startapp orders
```

پرونده settings.py پروژه خود را ویرایش کرده و برنامه جدید را به تنظیمات INSTALLED_APPS به شرح زیر اضافه کنید:

```
INSTALLED_APPS = [
    # ...
    'orders.apps.OrdersConfig',
]
```

شما برنامه سفارشات را فعال کرده اید.

[ایجاد مدل های سفارش](#)

برای ذخیره جزئیات سفارش به یک مدل نیاز دارید و یک مدل دیگر برای ذخیره کالاهای خریداری شده از جمله قیمت و کمیت آنها. پرونده model.py برنامه سفارشات را ویرایش کنید و کد زیر را به آن اضافه کنید:

```
from django.db import models
from shop.models import Product

class Order(models.Model):
    first_name = models.CharField(max_length=50)
    last_name = models.CharField(max_length=50)
    email = models.EmailField()
    address = models.CharField(max_length=250)
    postal_code = models.CharField(max_length=20)
    city = models.CharField(max_length=100)
    created = models.DateTimeField(auto_now_add=True)
    updated = models.DateTimeField(auto_now=True)
    paid = models.BooleanField(default=False)

    class Meta:
        ordering = ('-created',)

    def __str__(self):
        return 'Order {}'.format(self.id)

    def get_total_cost(self):
        return sum(item.get_cost() for item in self.items.all())


class OrderItem(models.Model):
    order = models.ForeignKey(Order,
                              related_name='items',
                              on_delete=models.CASCADE)
    product = models.ForeignKey(Product,
                               related_name='order_items',
                               on_delete=models.CASCADE)
    price = models.DecimalField(max_digits=10, decimal_places=2)
    quantity = models.PositiveIntegerField(default=1)

    def __str__(self):

        return '{}'.format(self.id)

    def get_cost(self):
        return self.price * self.quantity
```

مدل Order شامل چندین زمینه برای ذخیره مشتری است

اطلاعات و یک میدان بولی پرداخت می شود ، که پیش فرض False است. بعداً ، ما قصد داریم از این زمینه برای تمایز بین سفارشات پرداخت شده و پرداخت نشده استفاده کنیم. ما همچنین یک متدهای get_total_cost() برای به دست آوردن هزینه کل کالاهای خریداری شده در این سفارش تعریف می کنیم.

مدل OrderItem به ما امکان ذخیره محصول ، کمیت و قیمت پرداخت شده برای هر مورد را می دهد. ما شامل get_cost() می شویم تا هزینه مورد را برگردانیم.

دستور بعدی را برای ایجاد مهاجرت اولیه برای استفاده از دستور اجرا کنید:

```
python manage.py makemigrations
```

خروجی زیر را مشاهده خواهید کرد:

```
Migrations for 'orders':  
  orders/migrations/0001_initial.py  
    - Create model Order  
    - Create model OrderItem
```

دستور زیر را برای اعمال مهاجرت جدید اجرا کنید:

```
python manage.py migrate
```

مدلهای سفارش شما اکنون در پایگاه داده همگام شده اند.

مدل های سفارش در سایت مدیریت

بایاید مدل های قدیمی را به سایت مدیریت اضافه کنیم. فایل admin.py را از دستور برنامه ویرایش کنید تا به این شکل ظاهر شود:

```
from django.contrib import admin
from .models import Order, OrderItem

class OrderItemInline(admin.TabularInline):
    model = OrderItem
    raw_id_fields = ['product']

@admin.register(Order)
class OrderAdmin(admin.ModelAdmin):
    list_display = ['id', 'first_name', 'last_name', 'email',
                    'address', 'postal_code', 'city', 'paid',
                    'created', 'updated']
    list_filter = ['paid', 'created', 'updated']
    inlines = [OrderItemInline]
```

ما از یک کلاس ModelAdmin برای مدل OrderItem استفاده می کنیم تا آن را به عنوان آن درج کنیم درون کلاس در کلاس OrderAdmin. درون خطی به شما امکان می دهد یک مدل را در همان صفحه ویرایش مدل مربوطه خود قرار دهید.

سرور توسعه را با دستور python management.py runserver اجرا کنید و سپس /http://127.0.0.1:8000/admin/order/order/add را در صفحه خود باز کنید.

مرورگر صفحه زیر را مشاهده خواهید کرد:

First name:	<input type="text"/>																				
Last name:	<input type="text"/>																				
Email:	<input type="text"/>																				
Address:	<input type="text"/>																				
Postal code:	<input type="text"/>																				
City:	<input type="text"/>																				
<input type="checkbox"/> Paid																					
ORDER ITEMS <table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th>PRODUCT</th> <th>PRICE</th> <th>QUANTITY</th> <th>DELETE?</th> </tr> </thead> <tbody> <tr> <td><input type="text"/></td> <td><input type="text"/></td> <td><input type="text"/></td> <td><input type="button" value="X"/></td> </tr> <tr> <td><input type="text"/></td> <td><input type="text"/></td> <td><input type="text"/></td> <td><input type="button" value="X"/></td> </tr> <tr> <td><input type="text"/></td> <td><input type="text"/></td> <td><input type="text"/></td> <td><input type="button" value="X"/></td> </tr> <tr> <td colspan="4"> + Add another Order item </td> </tr> </tbody> </table>		PRODUCT	PRICE	QUANTITY	DELETE?	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="button" value="X"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="button" value="X"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="button" value="X"/>	+ Add another Order item			
PRODUCT	PRICE	QUANTITY	DELETE?																		
<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="button" value="X"/>																		
<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="button" value="X"/>																		
<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="button" value="X"/>																		
+ Add another Order item																					
<input type="button" value="Save and add another"/> <input type="button" value="Save and continue editing"/> <input style="background-color: #007bff; color: white; border-radius: 5px; padding: 5px; font-weight: bold; border: none;" type="button" value="SAVE"/>																					

ایجاد سفارشات مشتری

هنگامی که کاربر در نهایت سفارش خود را قرار داد ، ما از مدلهای نظمی که ایجاد کرده ایم برای ماندگار کردن موارد موجود در سبد خرید استفاده خواهیم کرد. پس از این مراحل سفارش جدید ایجاد می شود:

- 1- برای پر کردن اطلاعات خود فرم سفارش را به کاربران ارائه دهید
2. با داده های وارد شده ، یک نمونه سفارش جدید ایجاد کنید و یک نمونه OrderItem همراه برای هر مورد موجود در سبد خرید ایجاد کنید

۳- کلیه محتوای سبد خرید را پاک کنید و کاربران را به صفحه موفقیت هدایت کنید ابتدا برای وارد کردن جزئیات سفارش ، به یک فرم نیاز داریم. یک پرونده جدید در فهرست برنامه های کاربردی سفارش ایجاد کرده و آن را form.py بنویسید.

کد زیر را به آن اضافه کنید:

```
from django import forms
from .models import Order

class OrderCreateForm(forms.ModelForm):
    class Meta:
        model = Order
        fields = ['first_name', 'last_name', 'email', 'address',
                  'postal_code', 'city']
```

این روشی است که ما می خواهیم از آن برای ایجاد اشیاء سفارش جدید استفاده کنیم. اکنون برای رسیدگی به فرم و ایجاد سفارش جدید ، به یک نمای نیاز داریم. فایل views.py برنامه سفارشات را ویرایش کنید و کد زیر را به آن اضافه کنید:

```
from django.shortcuts import render
from .models import OrderItem
from .forms import OrderCreateForm
from cart.cart import Cart
```

```

def order_create(request):
    cart = Cart(request)
    if request.method == 'POST':
        form = OrderCreateForm(request.POST)
        if form.is_valid():
            order = form.save()
            for item in cart:
                OrderItem.objects.create(order=order,
                                         product=item['product'],
                                         price=item['price'],
                                         quantity=item['quantity'])
            # clear the cart
            cart.clear()
            return render(request,
                          'orders/order/created.html',
                          {'order': order})
    else:
        form = OrderCreateForm()
    return render(request,
                  'orders/order/create.html',
                  {'cart': cart, 'form': form})

```

در نمای Order_create ، سبد خرید فعلی را از جلسه با (cart = Cart(request)) دریافت می کنیم. بسته به روش درخواست ، کارهای زیر را انجام خواهیم داد:

درخواست GET: فرم OrderCreateForm را مرتب می کند و الگوی orders/order/create.html را ارائه می دهد.

درخواست ارسال: داده های ارسال شده در درخواست را تأیید می کند. اگر داده ها معتبر باشند ، ما با استفاده از دستور = () سفارش جدیدی را در دیتابیس ایجاد می کنیم. ما روی موارد سبد خرید تکرار می کنیم و برای هر یک از آنها OrderItem ایجاد می کنیم. در آخر ، محتوای سبد خرید را پاک می کنیم و دستورات orders/order/created.html را اجرا می کنیم.

یک پرونده جدید در فهرست برنامه های کاربردی سفارش ایجاد کرده و نام آن را urls.py کنید: urls.py

```
from django.urls import path
```

```
from . import views

app_name = 'orders'

urlpatterns = [
    path('create/', views.order_create, name='order_create'),
]
```

این الگوی URL برای مشاهده سفارش_ساخته است. پرونده urls.py را ویرایش کنید: myshop و شامل الگوی زیر است. به یاد داشته باشید که آن را قبل از الگوی shop.urls قرار دهید:

```
path('orders/', include('orders.urls', namespace='orders')),
```

قالب cart/detail.html برنامه سبد خرید را ویرایش کنید و این خط را ویرایش کنید:

```
<a href="#" class="button">Checkout</a>
```

URL_RCreate URL را به شرح زیر اضافه کنید:

```
<a href="{% url "orders:order_create" %}" class="button">  
    Checkout  
</a>
```

کاربران هم اکنون می توانند از صفحه جزئیات سبد خرید به فرم سفارش حرکت کنند.

ما هنوز باید قالب هایی را برای قرار دادن سفارشات تعریف کنیم. ساختار پرونده زیر را در فهرست برنامه کاربردی سفارشات ایجاد کنید:

```
templates/  
    orders/  
        order/  
            create.html  
            created.html
```

الگوی orders/order/create.html را ویرایش کنید و کد زیر را وارد کنید:

```

{% extends "shop/base.html" %}

{% block title %}
    Checkout
{% endblock %}

{% block content %}
<h1>Checkout</h1>

<div class="order-info">
    <h3>Your order</h3>
    <ul>
        {% for item in cart %}
            <li>
                {{ item.quantity }}x {{ item.product.name }}
                <span>${{ item.total_price }}</span>
            </li>
        {% endfor %}
    </ul>
    <p>Total: ${{ cart.get_total_price }}</p>
</div>

<form action"." method="post" class="order-form">
    {{ form.as_p }}
    <p><input type="submit" value="Place order"></p>
    {% csrf_token %}
</form>
{% endblock %}

```

در این قالب موارد سبد خرید از جمله کل و فرم برای نمایش سفارش نمایش داده می شود.

الگوی orders/order/created.html را ویرایش کنید و کد زیر را اضافه کنید:

```
{% extends "shop/base.html" %}

{% block title %}
    Thank you
{% endblock %}

{% block content %}
    <h1>Thank you</h1>
    <p>Your order has been successfully completed. Your order number is
    <strong>{{ order.id }}</strong>.</p>
{% endblock %}
```

این الگوی است که هنگام سفارش با موفقیت ایجاد می شود.

برای پیگیری پرونده های جدید ، سرور توسعه وب را راه اندازی کنید. <http://127.0.0.1:8000/> را در مرورگر خود باز کنید ، دو محصول را به سبد خرید اضافه کنید و به صفحه پرداخت ادامه دهید. صفحه ای مانند صفحه زیر را مشاهده خواهید کرد:

Checkout

First name:

Last name:

Email:

Address:

Postal code:

City:

Place order

Your order

- | | |
|-----------------|--------|
| • 1x Tea powder | \$21.2 |
| • 2x Red tea | \$91.0 |

Total: \$112.2

فرم را با داده های معتبر پر کنید و بر روی دکمه مکان سفارش کلیک کنید. ترتیب ایجاد می شود و صفحه موفقیتی را مانند این مشاهده خواهید کرد:

اکنون ، به سایت مدیریت بروید.

انجام کارهای ناهمزمان با Celery

همه آنچه شما در یک اجرای اجرا می کنید ، بر زمان پاسخ تأثیر می گذارد. در بسیاری از مواقع ، شما ممکن است بخواهید در اسرع وقت پاسخی را به کاربر برگردانید و بگذارید سرور بطور غیرمستقیم برخی فرآیندها را انجام دهد. این امر به ویژه برای فرآیندهای وقت گیر یا فرآیندهای در معرض شکست ، که ممکن است به یک سیاست آزمایش مجدد نیاز داشته باشد ، اهمیت دارد. به عنوان مثال ، یک پلت فرم به اشتراک گذاری ویدیو به کاربران امکان می دهد فیلم ها را بارگذاری کنند اما برای رمزگذاری فیلم های آپلود شده به مدت طولانی نیاز دارد. این سایت ممکن است پاسخی را به کاربران بازگرداند تا به آنها اطلاع دهد که کدگذاری به زودی آغاز خواهد شد و کد نویسی همزمان را به صورت غیر همزمان آغاز می کنید. نمونه دیگر ارسال ایمیل به کاربران است. اگر سایت شما اعلان های ایمیل را از یک نمایه ارسال می کند ، اتصال SMTP ممکن است پاسخ را خراب یا کند کند. برای جلوگیری از انسداد اجرای کد ، راه اندازی کارهای غیر همزمان ضروری است.

کرفس یک صفت کار توزیع شده است که می تواند تعداد زیادی پیام را پردازش کند. این پردازش را در زمان واقعی انجام می دهد اما همچنین از برنامه ریزی کار پشتیبانی می کند. با استفاده از کرفس نه تنها می توانید کارهای آسنکرون را به راحتی ایجاد کنید و به آنها اجازه دهید در اسرع وقت توسط کارگران اعدام شوند ، بلکه می توانید آنها را نیز برای برنامه ریزی در یک زمان خاص برنامه ریزی کنید.

نصب Celery

بایاید Celery را نصب کنیم و آن را در پروژه خود ادغام کنیم. کرفس را از طریق پیپ با استفاده از دستور زیر نصب کنید:

```
| pip install celery:
```

برای رسیدگی به درخواست های یک منبع خارجی به یک کارگزار پیام نیاز دارد. کارگزار مراقبت از ارسال پیام به کارگران Celery را انجام می دهد که وظایف آنها را هنگام دریافت آنها پردازش می کند. بایاید یک کارگزار پیام نصب کنیم.

نصب RabbitMQ

گزینه های مختلفی برای انتخاب به عنوان یک کارگزار پیام برای Celery وجود دارد ، از جمله فروشگاه های مهم و کلیدی مانند Redis یا یک سیستم پیام واقعی مانند RabbitMQ. ما کرس را با RabbitMQ پیکربندی می کنیم ، زیرا این کارگر پیام توصیه شده برای Celery است.

اگر از لینوکس استفاده می کنید ، می توانید RabbitMQ را از طریق پوسته با استفاده از دستور زیر نصب کنید:

```
apt-get install rabbitmq
```

اگر نیاز به نصب RabbitMQ در macOS یا ویندوز دارید ، می توانید نسخه های مستقل را در <https://www.rabbitmq.com/download.html> پیدا کنید.

پس از نصب آن ، RabbitMQ را با استفاده از دستور زیر از پوسته راه اندازی کنید:

```
rabbitmq-server
```

خروجی را خواهید دید که با خط زیر پایان می یابد:

```
Starting broker... completed with 10 plugins.
```

RabbitMQ در حال اجرا و آماده دریافت پیام است.

celery را به پروژه خود اضافه کنید

شما باید یک نمونه پیکربندی را برای نمونه Celery تهیه کنید. پرونده جدیدی را در کنار پرونده setting.py فروشگاه من ایجاد کنید و آن را celery.py بنامید. این پرونده شامل پیکربندی کرفس برای پروژه شما خواهد بود. کد زیر را به آن اضافه کنید:

```
import os
from celery import Celery

# set the default Django settings module for the 'celery' program.
os.environ.setdefault('DJANGO_SETTINGS_MODULE', 'myshop.settings')

app = Celery('myshop')

app.config_from_object('django.conf:settings', namespace='CELERY')
app.autodiscover_tasks()
```

در این کد موارد زیر را انجام می دهیم:

1. متغیر DJANGO_SETTINGS_MODULE را برای برنامه خط فرمان Celery تنظیم می کنیم.
2. ما نمونه ای از برنامه را با برنامه =کرفس ("myshop") ایجاد می کنیم.
3. ما هر پیکربندی سفارشی را از تنظیمات پروژه خود با استفاده از روش config_from_object () بارگیری می کنیم. ویژگی namespace پیشوندی را که تنظیمات مربوط به Celery در پرونده تنظیمات ما دارد ، مشخص می کند. با تنظیم فضای نام CELERY ، همه تنظیمات Celery باید پیشوند _CELERY_ را به نام خود درج کنند (به عنوان مثال ، (CELERY_BROKER_URL
- 4- سرانجام ، ما به کلر می گوییم که کارهای ناهمzman را برای برنامه های خود به طور خودکار کشف کند. کرفس به منظور بارگیری کارهای ناهمzman تعریف شده در آن ، به دنبال یک فایل task.py در هر فهرست برنامه کاربردی برنامه های اضافه شده به INSTALLED_APPS است.

شما نیاز به وارد کردن مازول کرفس در پرونده __init__.py پروژه خود دارید تا مطمئن شوید که هنگام شروع جنگو بارگیری شده است. پرونده myshop / __init__.py را ویرایش کنید و کد زیر را به آن اضافه کنید:

```
# import celery
from .celery import app as celery_app
```

اکنون می توانید برنامه نویسی کارهای غیر همزمان را برای برنامه های خود شروع کنید.

وظایف ناهمزمان را به برنامه خود اضافه کنید

ما می خواهیم یک کار ناهمزمان ایجاد کنیم تا هنگام سفارش ، یک اعلان ایمیل را برای کاربران خود ارسال کنیم. این کنوانسیون شامل وظایف ناهمزمان برای برنامه شما در یک مژوول وظایف در فهرست برنامه شما است.

یک پرونده جدید را درون برنامه سفارش ایجاد کرده و نام آن را task.py بنویسید این مکانی است که celery به دنبال کارهای ناهم زمان است. کد زیر را به آن اضافه کنید:

ما با استفاده از دکوراتور وظیفه `render_created` را تعریف می کنیم. همانطور که مشاهده می کنید ، یک کارکرفس فقط یک کارکرد پایتون است که با کار تزئین شده است.

عملکرد وظیفه ما یک پارامتر `order_id` را دریافت می کند. همیشه توصیه می شود هنگام انجام کار ، فقط شناسه ها را به توابع وظیفه و جستجوی اشیاء جستجو کنید. ما از تابع `send_mail()` تدارک دیده شده توسط Django استفاده می کنیم تا یک اعلان نامه الکترونیکی را برای کاربر ارسال کننده سفارش ارسال کنیم.

شما یاد گرفتید که چگونه پیکربندی Django را برای استفاده از سرور SMTP خود در فصل 2 ، تقویت و بلاگ خود با ویژگی های پیشرفته انجام دهید. اگر نمی خواهید تنظیمات ایمیل را تنظیم کنید ، می توانید با افزودن تنظیمات زیر به پرونده تنظیمات ایمیل به کنسول بنویسید:

```
EMAIL_BACKEND = 'django.core.mail.backends.console.EmailBackend'
```

حال باید وظیفه خود را به `order_create` اضافه کنیم. پرونده `views.py` برنامه سفارش را ویرایش کنید ، این کار را وارد کنید و پس از پاک کردن سبد خرید به شرح زیر با کار `order_created_asynchronous` تماس بگیرید:

```
from .tasks import order_created

def order_create(request):
    # ...
    if request.method == 'POST':
        # ...
        if form.is_valid():
            # ...
            cart.clear()
            # launch asynchronous task
            order_created.delay(order.id)
        # ...
```

ما `delay()` را برای اجرای آن به صورت غیر همزمان می نامیم.

این کار به صفت اضافه می شود و در اسرع وقت توسط یک کارگر اجرا می شود.

با استفاده از دستور زیر ، پوسته دیگری را باز کرده و celery worker را از فهرست پروژه خود شروع کنید:

```
celery -A myshop worker -l info
```

اکنون در حال اجرا و آماده پردازش وظایف است. اطمینان حاصل کنید که سرور توسعه Django نیز در حال اجرا است. <http://127.0.0.1:8000/> را در مرورگر خود باز کنید ، برخی از محصولات را به سبد خرید خود اضافه کنید و یک سفارش را تکمیل کنید. در پوسته ، شما celery worker را شروع کردید و خروجی شبیه به این را خواهید دید:

```
[2017-12-17 17:43:11,462: INFO/MainProcess] Received task:  
orders.tasks.order_created[e990ddae-2e30-4e36-b0e4-78bbd4f2738e]  
[2017-12-17 17:43:11,685: INFO/ForkPoolWorker-4] Task  
orders.tasks.order_created[e990ddae-2e30-4e36-b0e4-78bbd4f2738e] succeeded in  
0.22019841300789267s: 1
```

کار انجام شده است و برای سفارش خود یک اعلان نامه الکترونیک دریافت خواهد کرد.

نظرارت بر celery

ممکن است بخواهید وظایف ناهمزن اجرا شده را کنترل کنید. گل flower ابزاری مبتنی بر وب برای نظرارت بر celery است. با استفاده از این دستور می توانید Flower را نصب کنید:

```
pip install flower
```

پس از نصب ، می توانید با اجرای دستور زیر از فهرست پروژه خود ، Flower را راه اندازی کنید:

```
celery -A myshop flower
```

دارشبورد را در مرورگر خود باز کنید. شما قادر خواهید بود celery workers فعال و آمار وظایف ناهم زمان را مشاهده کنید:

The screenshot shows the Flower web application interface. At the top, there is a navigation bar with tabs: Flower (selected), Dashboard, Tasks, Broker, Monitor, Logout, Docs, and Code. Below the navigation bar, there is a summary section with five status indicators: Active: 0, Processed: 0, Failed: 0, Succeeded: 0, and Retried: 0. Underneath this is a search bar labeled "Search". A table displays worker information. The columns are: Worker Name, Status, Active, Processed, Failed, Succeeded, Retried, and Load Average. One worker entry is shown: celery@MacBook-Air-de-Antonio.local, Status: Online, Active: 0, Processed: 0, Failed: 0, Succeeded: 0, Retried: 0, Load Average: 22,24,236. At the bottom left of the main content area, it says "Showing 1 to 1 of 1 entries".

Worker Name	Status	Active	Processed	Failed	Succeeded	Retried	Load Average
celery@MacBook-Air-de-Antonio.local	Online	0	0	0	0	0	22,24,236

می توانید مستندات مربوط به flower را در <https://flower.readthedocs.io> پیدا کنید.

خلاصه

در این فصل ، شما یک برنامه فروشگاه اصلی ایجاد کرده اید. شما کاتالوگ محصول را ایجاد کرده اید و با استفاده از جلسات سبد خرید ایجاد کرده اید. شما یک پردازشگر متن سفارشی را پیاده سازی کرده اید تا سبد خرید در دسترس قالب های خود قرار گیرد و یک فرم برای قرار دادن سفارشات ایجاد کنید. شما همچنین آموخته اید که چگونه کارهای ناهمزمان را با کرفس شروع کنید.

در فصل بعد یاد می گیرید که چگونه یک دروازه پرداخت را به فروشگاه خود ادغام کنید ، اقدامات سفارشی را به سایت مدیریت اضافه کنید ، داده ها را با فرمت CSV صادر کنید و فایل های PDF را بصورت پویا تولید کنید.

فصل هشتم

مدیریت پرداخت‌ها

و سفارشات

در فصل قبل ، شما یک فروشگاه آنلاین اساسی با یک فروشگاه کالا و یک سبد خرید ایجاد کردید. شما همچنین آموخته اید که چگونه کارهای ناهمزمان را با کرفس شروع کنید. در این فصل یاد می‌گیرید که چگونه یک دروازه پرداخت را در سایت خود ادغام کنید تا به کاربران اجازه دهدید با کارت اعتباری پرداخت کنند. شما همچنین می‌توانید سایت مدیریت را برای صادرات سفارشات به فرمت CSV گسترش دهید و فاکتورهای PDF تولید خواهید کرد.

در این فصل یاد می‌گیرید که:

یک درگاه پرداخت را در پروژه خود ادغام کنید

تصادرات سفارشات به پرونده‌های CSV

ایجاد نمای سفارشی برای سایت مدیریت

فاکتورهای PDF را بصورت پویا تولید کنید

یکپارچگی دروازه پرداخت

یک دروازه پرداخت به شما امکان می دهد تا پرداخت های آنلاین را پردازش کنید. با استفاده از یک درگاه پرداخت می توانید سفارشات مشتری را مدیریت کرده و پردازش پرداخت را به یک شخص ثالث مطمئن و مطمئن واگذار کنید. دیگر لازم نیست نگران پردازش کارتهای اعتباری در سیستم خود باشید.

چندین ارائه دهنده دروازه پرداخت برای انتخاب وجود دارد. ما قصد داریم به ادغام Braintree پردازیم ، که توسط سرویس های آنلاین محبوب مانند Airbnb یا Uber استفاده می شود. Braintree یک API را فراهم می کند که به شما امکان می دهد تا پرداخت های آنلاین را با روش های پرداخت چندگانه مانند کارت اعتباری ، PayPal و Apple Pay و Android Pay پردازش کنید.

می توانید اطلاعات بیشتری در مورد Braintree در <https://www.braintreepayments.com> کسب کنید.

Braintree گزینه های مختلف ادغام را ارائه می دهد. ساده ترین ادغام Drop-in است که شامل یک فرم پرداخت از پیش آمده شده است. با این حال ، به منظور سفارشی کردن رفتار و تجربه پرداخت ما ، قصد داریم از ادغام پیشرفته Hosted Fields استفاده کنیم. می توانید اطلاعات بیشتر در مورد ادغام زمینه های میزبانی شده را در <https://developers.braintreepayments.com/guides/hosted-fields/overview/javascript/v3> کسب کنید.

برخی از زمینه های پرداخت در صفحه پرداخت ، مانند شماره کارت اعتباری ، شماره CVV یا تاریخ انقضا باید به صورت iframe میزبانی شوند. یکپارچه سازی Hosted Fields میزبان زمینه های پرداخت در دامنه دروازه پرداخت است و یک iframe برای ارائه زمینه ها به کاربران ارائه می دهد. این امکان را برای شما فراهم می کند تا بتوانید ظاهر و نوع فرم پرداخت را شخصی سازی کنید ، ضمن اینکه از عدم مطابقت با الزامات صنعت کارت پرداخت (PCI) استفاده می کنید. از آنجا که می توانید ظاهر و شکل زمینه های فرم را به صورت دلخواه تنظیم کنید ، کاربران متوجه iframe نخواهند شد.

ایجاد یک حساب کاربری sandbox Braintree

برای ادغام دروازه پرداخت به سایت خود ، به یک حساب Braintree نیاز دارید. باید برای آزمایش API یک حساب sandbox ایجاد کنیم.

برای ایجاد یک حساب جدید sandbox جزئیات را پر کنید. برای تکمیل تنظیم حساب خود یک ایمیل از Braintree

خواهید کرد:

Test Everything Braintree

Entering our sandbox allows you to get a feel for the Braintree experience before applying for a merchant account or going to production.

Already in the sandbox? [Sign in.](#)

Sign up for the sandbox

Full name

First name Last name

Company name

Where is your business located?

Spain

Email address

me@example.com

Try the sandbox

برای ایجاد یک حساب جدید sandbox جزئیات را پر کنید. برای تکمیل تنظیم حساب خود یک ایمیل از Braintree دریافت خواهید کرد.

پیوند را دنبال کنید و تنظیم حساب خود را تکمیل کنید. پس از اتمام کار ، در <https://sandbox.braintreegateway.com/login> وارد شوید. شناسه تجاری شما و کلیدهای private / public به این صورت نمایش داده می شوند:

Sandbox Keys & Configuration

Here are the keys to your Sandbox Account. Once you're ready to start taking payments with a production Braintree Account you'll have to update your code, replacing these with your production Braintree Account keys.

Merchant ID: 9xtfhm7sv733jznk

Public Key: q8fxx6fwkjx8dfkw

برای تأیید صحت درخواست های Braintree API به این اطلاعات نیاز دارید. همیشه کلید خصوصی را مخفی نگه دارید.

نصب ماژول Braintree Python

Braintree ماژول پایتون را فراهم می کند که برخورد با API آن را ساده می کند. کد منبع در https://github.com/braintree/braintree_python قرار دارد. ما قصد داریم با استفاده از ماژول دروازه پرداخت را در پروژه خود ادغام کنیم.

با استفاده از دستور زیر ، ماژول braintree را از پوسته نصب کنید:

```
pip install braintree==3.45.0
```

تنظیمات زیر را به پرونده settings.py پروژه خود اضافه کنید:

```
# Braintree settings
BRAINTREE_MERCHANT_ID = 'XXX' # Merchant ID
BRAINTREE_PUBLIC_KEY = 'XXX' # Public Key
BRAINTREE_PRIVATE_KEY = 'XXX' # Private key

from braintree import Configuration, Environment

Configuration.configure(
    Environment.Sandbox,
    BRAINTREE_MERCHANT_ID,
    BRAINTREE_PUBLIC_KEY,
    BRAINTREE_PRIVATE_KEY
)
```

مقادیر BRAINTREE_PRIVATE_KEY و BRAINTREE_PUBLIC_KEY، BRAINTREE_MERCHANT_ID را با موارد حساب خود جایگزین کنید.

باید دروازه پرداخت را در فرایند پرداخت هزینه ادغام کنیم.

پکارچه کردن دروازه پرداخت

مراحل پرداخت به شرح زیر خواهد بود:

1. موارد را به سبد خرید اضافه کنید
2. سبد خرید را بررسی کنید
3. جزئیات کارت اعتباری را وارد کرده و پرداخت کنید

ما قصد داریم یک برنامه جدید برای مدیریت پرداخت ها ایجاد کنیم. با استفاده از دستور زیر یک برنامه جدید در پروژه خود

ایجاد کنید:

```
python manage.py startapp payment
```

پرونده settings.py پروژه خود را ویرایش کرده و برنامه جدید را به تنظیمات INSTALLED_APPS به شرح زیر اضافه کنید:

```
INSTALLED_APPS = [
    # ...
    'payment.apps.PaymentConfig',
]
```

برنامه پرداخت اکنون فعال است.

بعد از اینکه مشتری سفارش می دهد ، باید آنها را به روند پرداخت هدایت کنیم. پرونده views.py درخواست سفارش را ویرایش کنید و واردات زیر را نیز شامل شوید:

```
from django.urls import reverse
```

```
from django.shortcuts import render, redirect
```

در همان پرونده ، خطوط زیر نمایش `order_create` را جایگزین کنید:

```
# launch asynchronous task
order_created.delay(order.id)
return render(request,
              'orders/order/created.html',
              locals())
```

آنها را با موارد زیر جایگزین کنید:

```
# launch asynchronous task
order_created.delay(order.id)
# set the order in the session
request.session['order_id'] = order.id
# redirect for payment
return redirect(reverse('payment:process'))
```

با استفاده از این کد ، پس از ایجاد موفقیت در سفارش ، شناسه سفارش را در جلسه فعلی با استفاده از کلید جلسه `order_id` تنظیم می کنیم. سپس ، کاربر را به پرداخت هدایت می کنیم: URL فرآیند ، که می خواهیم بعداً آن را پیاده سازی کنیم.

بخاطر داشته باشید که برای سفارش و اجرای دستور ، باید celery را اجرا کنید.

هر بار سفارش در Braintree ایجاد می شود ، یک شناسه معاملات منحصر به فرد ایجاد می شود. برای ذخیره شناسه معاملات ، ما یک قسمت جدید به مدل سفارش برنامه سفارش اضافه می کنیم. این به ما امکان می دهد تا هر سفارش را با معاملات مربوط به Braintree خود پیوند دهیم.

پرونده `model.py` برنامه سفارش را ویرایش کنید و قسمت زیر را به مدل `Order` اضافه کنید:

```
class Order(models.Model):
    ...
    braintree_id = models.CharField(max_length=150, blank=True)
```

باید این قسمت را با بانک اطلاعات همگام سازی کنیم. برای ایجاد مهاجرت از دستور زیر استفاده کنید:

```
python manage.py makemigrations
```

خروجی زیر را مشاهده خواهید کرد:

```
Migrations for 'orders':
  orders/migrations/0002_order_braintree_id.py
    - Add field braintree_id to order
```

مهاجرت را با دستور زیر به پایگاه داده اعمال کنید:

```
python manage.py migrate
```

خروجی را خواهید دید که با خط زیر پایان می یابد:

```
Applying orders.0002_order_braintree_id... OK
```

تغییرات مدل اکنون با بانک اطلاعات همگام شده است. اکنون می توانید شناسه معاملات Braintree را برای هر سفارش ذخیره کنید. باید دروازه پرداخت را ادغام کنیم.

ادغام Braintree با استفاده از زمینه های میزبانی شده

ادغام Hosted Fields به شما امکان می دهد فرم پرداخت خود را با استفاده از سبک ها و طرح های سفارشی ایجاد کنید. iframe با استفاده از Braintree JavaScript SDK به صورت پویا به صفحه اضافه می شود. شامل فرم پرداخت Hosted Fields می باشد. هنگامی که مشتری فرم را ارسال می کند ، Hosted Fields اطلاعات کارت را به صورت این مجموع می کند و سعی می کند تا آنها را علامت گذاری کند. اگر رمزگذاری موفقیت آمیز باشد ، می توانید کد تولید نشانه را به نمای خود ارسال کنید تا با استفاده از ماژول braintree پایتون معامله انجام دهید.

ما نمایشی را برای پردازش پرداخت ها ایجاد خواهیم کرد. کل مراحل پرداخت به شرح زیر است:

1. در نمای ، یک توکن مشتری با استفاده از ماژول braintree Python تولید می شود. این نشانه در مرحله بعدی برای سریع کردن مشتری Braintree JavaScript استفاده می شود. این نشانه پرداخت نیست.

2. نمای الگوی پرداخت را ارائه می دهد. این قالب با استفاده از نشانه مشتری ، Braintree JavaScript SDK را باگیری می کند و در قسمت فرم پرداخت پرداخت میزبان ، iframe را تولید می کند.

3. کاربران جزئیات کارت اعتباری خود را وارد کرده و فرم را ارسال می کنند. یک شناسه پرداخت با مشتری Braintree ایجاد می شود. ما با درخواست POST نشانه را به نمای خود ارسال می کنیم.

4- نمای پرداخت نشانه را دریافت می کند و ما از آن برای تولید تراکنش با استفاده از ماژول braintree Python استفاده می کنیم.

بایاید با نمای پرداخت پرداخت شروع کیم. برنامه views.py پرونده نشانه پرداخت را ویرایش کنید و کد زیر را به آن اضافه کنید:

```
import braintree
from django.shortcuts import render, redirect, get_object_or_404
from orders.models import Order

def payment_process(request):
    order_id = request.session.get('order_id')
    order = get_object_or_404(Order, id=order_id)

    if request.method == 'POST':
        # retrieve nonce
        nonce = request.POST.get('payment_method_nonce', None)
        # create and submit transaction
        result = braintree.Transaction.sale({
            'amount': '{:.2f}'.format(order.get_total_cost()),
            'payment_method_nonce': nonce,
            'options': {
                'submit_for_settlement': True
            }
        })
        if result.is_success:
            # mark the order as paid
            order.paid = True
            # store the unique transaction id
            order.braintree_id = result.transaction.id
            order.save()
            return redirect('payment:done')
        else:
            return redirect('payment:canceled')
    else:
        # generate token
        client_token = braintree.ClientToken.generate()
        return render(request,
                      'payment/process.html',
                      {'order': order,
                       'client_token': client_token})
```

نمای pay_process فرایند پرداخت را مدیریت می کند. در این نظر اقدامات زیر را انجام دهید:

1. ما ترتیب فعلی را از کلید جلسه علمی rend_id ، که قبلاً در نمای_شکل تنظیم شده بود ، دریافت می کنیم.

2. ما شیء سفارش را برای شناسه داده شده بازیابی می کنیم یا در صورت عدم یافتن خطای 404 یافت نشد.

3. وقتی نمای با درخواست POST بارگیری می شود ، ما pay_method_nonce را برای بازیابی یک معامله جدید با استفاده از braintree.Transaction.sale() به دست می آوریم. پارامترهای زیر را به آن منتقل می کنیم:

1. مقدار: مبلغ کل برای شارژ مشتری.

2. نشانه توجیحی است که توسط Braintree برای پرداخت ایجاد شده است. با استفاده از Braintree JavaScript SDK در قالب تولید می شود.

3. گزینه ها: ما گزینه Sub_for_settlement را با True ارسال می کنیم تا معامله به طور خودکار برای تسویه حساب ارسال شود.

4- اگر معامله با موفقیت پرداش شد ، ما با تنظیم کردن ویژگی پرداخت شده آن به True ، علامت پرداخت شده را علامت گذاری می کنیم و شناسه معامله منحصر به فرد را که توسط دروازه به آن برگردانده شده است در صفت braintree_id ذخیره می کنیم. ما کاربر را به پرداخت هدایت می کنیم: اگر پرداخت در غیر این صورت به پرداخت موفقیت آمیز باشد ، URL را انجام داد: لغو شد.

5- اگر نمای با درخواست GET بارگذاری شد ، ما یک نشان مشتری را ایجاد می کنیم که در قالب استفاده می کنیم تا مشتری Braintree JavaScript بیاید در صورت موفقیت آمیز بودن پرداخت یا وقتی که به هر دلیلی پرداخت شده است ، نظرات اولیه را برای تغییر مسیر کاربران ایجاد کنیم. کد زیر را به پرونده views.py برنامه پرداخت اضافه کنید:

```
def payment_done(request):
    return render(request, 'payment/done.html')

def payment_canceled(request):
    return render(request, 'payment/canceled.html')
```

یک پرونده جدید را در فهرست راهنمای برنامه پرداخت ایجاد کنید و نام آن را urls.py دهید. کد زیر را به آن اضافه کنید:

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

app_name = 'payment'

urlpatterns = [
    path('process/', views.payment_process, name='process'),
    path('done/', views.payment_done, name='done'),
    path('canceled/', views.payment_canceled, name='canceled'),
]
```

این URL ها برای گردش کار پرداخت هستند. ما الگوهای URL زیر را نجانده ایم:

پردازش: نمایی که پرداخت را پردازش می کند

انجام شده: نمای جهت تغییر مسیر کاربر در صورت موفقیت آمیز بودن پرداخت

لغو شده است: نمای جهت تغییر مسیر کاربر در صورت عدم موفقیت آمیز بودن پرداخت

پرونده urls.py اصلی پروژه myshop را ویرایش کنید و الگوهای URL برای برنامه پرداخت را به شرح زیر وارد کنید:

```
urlpatterns = [
    # ...
    path('payment/', include('payment.urls', namespace='payment')),
    path('', include('shop.urls', namespace='shop')),
]
```

به یاد داشته باشید که برای جلوگیری از تطابق الگوی ناخواسته، آن را قبل از الگوی shop.urls قرار دهید.

ساختار پرونده زیر را در فهرست راهنمای برنامه پرداخت ایجاد کنید:

```
templates/  
  payment/  
    process.html  
    done.html  
    canceled.html
```

الگوی پرداخت / payment را ویرایش کنید و کد زیر را به آن اضافه کنید:

```
{% extends "shop/base.html" %}

{% block title %}Pay by credit card{% endblock %}

{% block content %}
<h1>Pay by credit card</h1>
<form action"." id="payment" method="post">

    <label for="card-number">Card Number</label>
    <div id="card-number" class="field"></div>

    <label for="cvv">CVV</label>
    <div id="cvv" class="field"></div>

    <label for="expiration-date">Expiration Date</label>
    <div id="expiration-date" class="field"></div>

    <input type="hidden" id="nonce" name="payment_method_nonce" value="">
    {% csrf_token %}
    <input type="submit" value="Pay">
</form>
<!-- Load the required client component. -->
<script src="https://js.braintreegateway.com/web/3.29.0/js/client.min.js">
</script>
<!-- Load Hosted Fields component. -->
<script src="https://js.braintreegateway.com/web/3.29.0/js/hosted-fields.min.js"></script>
<script>
    var form = document.querySelector('#payment');
    var submit = document.querySelector('input[type="submit"]');

```

```
braintree.client.create({
  authorization: '{{ client_token }}'
}, function (clientErr, clientInstance) {
  if (clientErr) {
    console.error(clientErr);
    return;
  }

  braintree.hostedFields.create({
    client: clientInstance,
    styles: {
      'input': {'font-size': '13px'},
      'input.invalid': {'color': 'red'},
      'input.valid': {'color': 'green'}
    },
    fields: {
      number: {selector: '#card-number'},
      cvv: {selector: '#cvv'},
      expirationDate: {selector: '#expiration-date'}
    }
  }, function (hostedFieldsErr, hostedFieldsInstance) {
    if (hostedFieldsErr) {
      console.error(hostedFieldsErr);
      return;
    }

    submit.removeAttribute('disabled');

    form.addEventListener('submit', function (event) {
      event.preventDefault();

      hostedFieldsInstance.tokenize(function (tokenizeErr, payload) {
        if (tokenizeErr) {
          console.error(tokenizeErr);
          return;
        }
        // set nonce to send to the server
        document.getElementById('nonce').value = payload.nonce;
        // submit form
        document.getElementById('payment').submit();
      });
    }, false);
  });
});
</script>
{% endblock %}
```

این الگوی است که فرم پرداخت را نشان می دهد و پرداخت را پردازش می کند. ظروف <div> را به جای عناصر <input> برای قسمتهای ورودی کارت اعتباری تعریف می کنیم: شماره کارت اعتباری ، شماره CVV و تاریخ انقضا. به این ترتیب زمینه هایی را تعیین می کنیم که مشتری Braintree JavaScript SDK client.min.js را شامل می شویم که برای ارسال یک نشانه توکن به نمای ما یک بار توسط مشتری Payment_method_nonce تولید می شود.

در الگوی خود ، ما Hosted Fields hosting -ields.min.js و مؤلفه Braintree JavaScript SDK client.min.js با رگذاری می کنیم. سپس کد جاوا اسکریپت زیر را اجرا می کنیم:

1. ما مشتری Braintree JavaScript SDK client.min.js را با استفاده از روش braintree.client.create() با استفاده از klient_token تولید شده توسط نمای Payment_process ، سریعاً بلافصله قرار می دهیم.
2. ما با استفاده از متده استفاده Hosted Fields braintree.hostedFields.create کامپونت را model می کنیم.
3. ما سبک های CSS سفارشی را برای قسمت های ورودی مشخص می کنیم.
- 4- انتخاب کننده شناسه را برای قسمتها مشخص می کنیم: cardnumber ، cvv و تاریخ انقضا.
- 5- ما برای عملکرد ارسال فرم ، یک شنونده رویداد را اضافه می کنیم.

هنگامی که فرم ارسال شد ، فیلدها با استفاده از Braintree SDK نشانه گذاری می شوند و نشانه توکن در قسمت تنظیم می شود. سپس فرم ارسال می شود تا نمای ما از پردازش پرداخت مبلغی دریافت کند. قالب payment/done.html را ویرایش کنید و کد زیر را به آن اضافه کنید:

```
{% extends "shop/base.html" %}

{% block content %}
    <h1>Your payment was successful</h1>
```

```
<p>Your payment has been processed successfully.</p>
{% endblock %}
```

این الگوی برای صفحه ای است که کاربر به دنبال پرداخت موفقیت آمیز هدایت می شود.

الگوی payment/canceled.html را تغییر دهید و کد زیر را به آن اضافه کنید:

```
{% extends "shop/base.html" %}

{% block content %}
    <h1>Your payment has not been processed</h1>
    <p>There was a problem processing your payment.</p>
{% endblock %}
```

این الگوی برای صفحه ای است که کاربر در صورت موفقیت آمیز بودن معامله به کاربر هدایت می شود. بیایید روند پرداخت را امتحان کنیم.

تست پرداختها

یک فرمان را باز کنید و با دستور زیر RabbitMQ را اجرا کنید:

```
rabbitmq-server
```

پوسته دیگری را باز کرده و celery worker را از فهرست پروژه خود با دستور زیر شروع کنید:

```
celery -A myshop worker -l info
```

یک پوسته دیگر را باز کنید و سرور توسعه را با این دستور شروع کنید:

```
python manage.py runserver
```

/http://127.0.0.1:8000 را در مرورگر خود باز کنید ، برخی از محصولات را به سبد خرید اضافه کنید و فرم پرداخت را پر کنید. با کلیک بر روی دکمه PLACE ORDER ، سفارش به بانک اطلاعات ادامه می یابد ، شناسه سفارش در جلسه فعلی ذخیره می شود و شما به صفحه فرآیند پرداخت هدایت می شوید.

صفحه فرآیند پرداخت ، سفارش را از جلسه بازیابی می کند و فرم Hosted Fields iframe را در قالب ارائه می دهد ، به

شرح زیر:

Pay by credit card

Card Number

CVV

Expiration Date

Pay

برای دیدن HTML تولید شده می توانید به کد منبع HTML نگاهی بیندازید.

Braintree لیستی از کارت‌های اعتباری موفق و ناموفق ارائه می دهد تا بتوانید تمام سناریوهای ممکن را آزمایش کنید. می توانید لیستی از کارت‌های اعتباری را برای آزمایش در <https://developers.braintreepayments.com/guides/cred> استفاده کنید. ما قصد داریم از کارت تست 1111 1111 1111 1111 VISA 4111 1111 1111 1111 استفاده کنیم که یک خرید موفق را برمی گرداند. ما قصد داریم از 123 CVV و هر تاریخ انقضا در آینده مانند 24/12 استفاده کنیم. جزئیات کارت اعتباری را به شرح زیر وارد کنید:

Pay by credit card

Card Number

4111 1111 1111 1111

CVV

123

Expiration Date

12 / 20

Pay

روی دکمه Pay کلیک کنید. صفحه زیر را مشاهده خواهید کرد:

My shop

Your payment was successful

Your payment has been processed successfully.

معامله با موفقیت پردازش شده است. اکنون می توانید در <https://sandbox.braintreegateway.com/login> به <https://braintreepayments.com> مراجعه کنید. حساب کاربری خود وارد شوید. تحت معاملات ، شما می توانید معامله مانند این را مشاهده کنید:

ID	Transaction Date	Type	Status	Customer	Payment Information	Amount
2bwkx5b6	02/05/2018 07:45:23 PM CST	Sale	Submitted For Settlement		 411111*****1111	21,20 € EUR

اکنون در مرورگر خود `/http://127.0.0.1:8000/admin/order/order` را باز کنید. اکنون باید سفارش به صورت پرداخت علامت گذاری شود و حاوی شناسه معاملات مربوط به Braintree باشد:

Paid

Braintree id:

2bwkx5b6

تبریک می گوییم! شما یک دروازه پرداخت را برای پردازش کارت‌های اعتباری پیاده سازی کرده اید.

زندگی می کنند

هنگامی که محیط خود را آزمایش کردید ، می توانید یک حساب Braintree واقعی در ایجاد کنید. پس از آماده شدن برای تولید ، یادتان باشد که زندگی خود را تغییر دهید

اعتبارنامه محیط در پرونده تنظیمات.py پروژه خود استفاده کنید و برای تنظیم محیط خود از استفاده کنید. کلیه مراحل پخش زنده در braintree.Engociation.Production خلاصه می شود. <https://developers.braintreepayments.com/start/guide-live/python>

صادرات سفارشات به پرونده های CSV

بعضی اوقات ، ممکن است بخواهید اطلاعات موجود در یک مدل را به یک پرونده صادر کنید تا بتوانید آن را در هر سیستم دیگری وارد کنید. یکی از فرمتهای پرکاربرد برای صادرات و واردات داده ، مقادیر جدا شده با کاما (CSV) است. پرونده CSV یک فایل متند ساده است که از تعدادی پرونده تشکیل شده است. معمولاً یک رکورد در هر خط وجود دارد ، و برخی از شخصیت های مشخص کننده ، معمولاً با کاما لفظی ، زمینه های ضبط را از هم جدا می کنند. ما می خواهیم سایت دولت را سفارشی کنیم تا بتوانیم سفارشات را به پرونده های CSV صادر کنیم.

افزودن اقدامات سفارشی به سایت مدیریت

جنگو انواع مختلفی از گزینه ها را برای شخصی سازی سایت مدیریت به شما ارائه می دهد. ما می خواهیم نمای لیست شی را اصلاح کنیم تا یک عملکرد مدیر سفارشی را شامل شود.

یک عملکرد سرپرست به شرح زیر عمل می کند: یک کاربر اشیاء را از صفحه لیست اشیاء سرپرست با کادرهای انتخاب انتخاب می کند ، سپس عملی را برای انجام در مورد همه موارد انتخاب کرده و عمل را اجرا می کند. تصویر زیر نشان می دهد که مکانها در محل مدیریت واقع شده اند:

Select user to change

The screenshot shows the Django Admin interface for managing users. At the top, there's a search bar with a magnifying glass icon and a 'Go' button. Below it, an 'Action:' dropdown is set to '—' and has a checkmark next to it. To the right, it says '0 of 1 selected'. There are two user entries listed: one with a checkbox labeled 'U...' and another labeled 'admin' with a checkbox. The 'admin' entry is highlighted with a blue background. A modal dialog box is overlaid on the page, centered over the action buttons, with the text 'Delete selected users' in white on a blue background.

می توانید با نوشتن یک عملکرد معمولی که پارامترهای زیر را دریافت می کند ، یک اقدام سفارشی ایجاد کنید:

فعلی نمایش داده می شود ModelAdmin

هدف درخواست فعلی به عنوان نمونه HttpRequest

برای اشیاء انتخاب شده توسط کاربر QuerySet

این عملکرد هنگامی انجام می شود که این اقدام از سایت مدیریت شروع شود.

ما قصد داریم یک اقدام مدیر سفارشی ایجاد کنیم تا لیستی از سفارشات به عنوان پرونده CSV بارگیری شود. فایل admin.py برنامه سفارش را ویرایش کنید و قبل از کلاس OrderAdmin کد زیر را اضافه کنید:

```

import csv
import datetime
from django.http import HttpResponseRedirect

def export_to_csv(modeladmin, request, queryset):
    opts = modeladmin.model._meta
    response = HttpResponseRedirect(content_type='text/csv')
    response['Content-Disposition'] = 'attachment;\'\n
        'filename={}.csv'.format(opts.verbose_name)
    writer = csv.writer(response)

    fields = [field for field in opts.get_fields() if not field.many_to_many\
and not field.one_to_many]
    # Write a first row with header information
    writer.writerow([field.verbose_name for field in fields])
    # Write data rows
    for obj in queryset:
        data_row = []
        for field in fields:
            value = getattr(obj, field.name)
            if isinstance(value, datetime.datetime):
                value = value.strftime('%d/%m/%Y')
            data_row.append(value)
        writer.writerow(data_row)
    return response
export_to_csv.short_description = 'Export to CSV'

```

در این کد کارهای زیر را انجام می دهیم:

1. ما نمونه ای از `HttpResponse` ، از جمله نوع متن / CSV متن سفارشی ایجاد می کنیم تا به مرورگر بگوییم که پاسخ باید به عنوان یک پرونده CSV درمان شود. ما همچنین یک هدر `Content-Disposition` اضافه می کنیم تا نشان دهد که پاسخ HTTP حاوی یک فایل پیوست شده است.
2. ما یک شیء نویسنده CSV ایجاد می کنیم که روی شیء پاسخ نوشته خواهد شد.
3. ما با استفاده از متد `get_fields()` از گزینه های مدل `meta` ، زمینه های مدل را به صورت پویا می گیریم. ما روابط خیلی تومنی و یک به یک را از هم جدا می کنیم.
- 4- ما یک ردیف سریگ شامل نام های فیلد را می نویسیم.

5. ما بیش از QuerySet داده شده تکرار می کنیم و برای هر شیء که توسط QuerySet برگردانده است ردیف می نویسیم.
ما از قالب بندی اشیاء datetime مراقبت می کنیم زیرا مقدار خروجی CSV باید یک رشته باشد.

6. با تنظیم یک ویژگی short_description به عملکرد ، نام نمایشگر را برای عمل در قالب سفارشی می کنیم.
ما یک عملکرد مدیر عمومی ایجاد کرده ایم که می تواند به هر کلاس ModelAdmin اضافه شود.
در آخر ، عملکرد جدید eksport_to_csv admin را به موارد زیر وارد کنید.

```
class OrderAdmin(admin.ModelAdmin):
    # ...
    actions = [export_to_csv]
```

. /http://127.0.0.1:8000/admin/order/order را در مرورگر خود باز کنید.

نتیجه عملکرد سرپرست باید به صورت زیر باشد:

Select order to change

Action: Export to CSV					
	ID	FIRST NAME	LAST NAME	EMAIL	ADDRESS
<input checked="" type="checkbox"/>	19	Antonio	Melé	antonio.mele@gmail.com	Bank Street
<input type="checkbox"/>	18	Django	Reinhardt	email@domain.com	Music Street

برخی از سفارشات را انتخاب کرده و از کادر انتخاب گزینه Export to CSV را انتخاب کرده و سپس بر روی دکمه GO کلیک کنید. مروگر شما فایل CSV تولید شده با نام order.csv را بارگیری می کند. پرونده بارگیری شده را با استفاده از ویرایشگر متن باز کنید. شما باید محتوای با قالب زیر را مشاهده کنید ، از جمله یک ردیف هدر و یک ردیف برای هر شیء سفارش که انتخاب کرده اید:

```
ID,first name,last name,email,address,postal  
code,city,created,updated,paid,braintree id  
3,Antonio,Melé,antonio.mele@gmail.com,Bank Street,WS  
J11,London,25/02/2018,25/02/2018,True,2bwkx5b6  
...  
...
```

همانطور که مشاهده می کنید ، ایجاد اقدامات سرپرست بسیار ساده است.

می توانید در مورد تولید پرونده های CSV با Django در اطلاعات بیشتری کسب کنید

[./https://docs.djangoproject.com/fa/2.0/howto/outputting-csv](https://docs.djangoproject.com/fa/2.0/howto/outputting-csv)

گسترش سایت سرپرست با نماهای سفارشی

بعضی اوقات ، ممکن است بخواهید از طریق پیکربندی ModelAdmin ، ایجاد اقدامات مدیریتی و نادیده گرفتن قالب های سرور ، سایت مدیریت را سفارشی کنید. اگر اینگونه باشد ، باید یک نمای سرپرست سفارشی ایجاد کنید. با یک نمای سفارشی ، شما می توانید هر کارکرد مورد نیاز خود را بسازید. فقط باید اطمینان حاصل کنید که فقط کاربران کارمندان می توانند به نمای شما دسترسی پیدا کنند و شما با ایجاد الگوی مدیریت و توسعه الگوی مدیریت ، نگاه و احساس مدیر را حفظ می کنید. بیایید یک نمایش سفارشی برای نمایش اطلاعات مربوط به یک سفارش ایجاد کنیم.

پرونده views.py برنامه سفارش را ویرایش کنید و کد زیر را به آن اضافه کنید:

```
from django.contrib.admin.views.decorators import staff_member_required
from django.shortcuts import get_object_or_404
from .models import Order

@staff_member_required
def admin_order_detail(request, order_id):
    order = get_object_or_404(Order, id=order_id)
    return render(request,
                  'admin/orders/order/detail.html',
                  {'order': order})
```

دکوراتور `Staff_member_required` برسی می کند که زمینه های `is_staff` و `is_active` کاربر درخواست کننده این صفحه روی `True` تنظیم شده است. در این نمای ، شیء سفارش را با شناسه داده شده دریافت می کنیم و یک الگوی ارائه می دهیم تا سفارش را نمایش دهیم.

اکنون پرونده `urls.py` برنامه سفارش را ویرایش کرده و الگوی URL زیر را به آن اضافه کنید:

```
path('admin/order/<int:order_id>', views.admin_order_detail,
      name='admin_order_detail'),
```

ساختار پرونده زیر را در `templates/ directory` برنامه سفارش ایجاد کنید:

```
admin/
  orders/
    order/
      detail.html
```

الگوی `detail.html` را ویرایش کنید و محتوای زیر را به آن اضافه کنید:

```
{% extends "admin/base_site.html" %}  
{% load static %}  
  
{% block extrastyle %}  
    <link rel="stylesheet" type="text/css" href="{% static "css/admin.css" %}"  
/>>  
{% endblock %}  
  
{% block title %}  
    Order {{ order.id }} {{ block.super }}  
{% endblock %}  
  
{% block breadcrumbs %}  
    <div class="breadcrumbs">  
        <a href="{% url "admin:index" %}">Home</a> &rsaquo;  
        <a href="{% url "admin:orders_order_changelist" %}">Orders</a>  
        &rsaquo;  
        <a href="{% url "admin:orders_order_change" order.id %}">Order {{  
order.id }}</a>  
        &rsaquo; Detail  
    </div>  
{% endblock %}  
  
{% block content %}  
    <h1>Order {{ order.id }}</h1>  
    <ul class="object-tools">  
        <li>  
            <a href="#" onclick="window.print();">Print order</a>  
        </li>  
    </ul>  
    <table>
```

```
<tr>
  <th>Created</th>
  <td>{{ order.created }}</td>
</tr>
<tr>
  <th>Customer</th>
  <td>{{ order.first_name }} {{ order.last_name }}</td>
</tr>
<tr>
  <th>E-mail</th>
  <td><a href="mailto:{{ order.email }}">{{ order.email }}</a></td>
</tr>
<tr>
  <th>Address</th>
  <td>{{ order.address }}, {{ order.postal_code }} {{ order.city }}</td>
</tr>
<tr>
  <th>Total amount</th>
  <td>${{ order.get_total_cost }}</td>
</tr>
<tr>
  <th>Status</th>
  <td>{%
    if order.paid %
    Paid
    {%
      else %
      Pending payment
    endif %
  }</td>
</tr>
</table>
```

```
<div class="module">
  <div class="tabular inline-related last-related">
    <table>
      <h2>Items bought</h2>
      <thead>
        <tr>
          <th>Product</th>
          <th>Price</th>
          <th>Quantity</th>
          <th>Total</th>
        </tr>
      </thead>
      <tbody>
        {% for item in order.items.all %}
        <tr class="row{% cycle "1" "2" %}">
          <td>{{ item.product.name }}</td>
          <td class="num">${{ item.price }}</td>
          <td class="num">{{ item.quantity }}</td>
          <td class="num">${{ item.get_cost }}</td>
        </tr>
        {% endfor %}
        <tr class="total">
          <td colspan="3">Total</td>
          <td class="num">${{ order.get_total_cost }}</td>
        </tr>
      </tbody>
    </table>
  </div>
</div>
[% endblock %]
```

این قالب برای نمایش جزئیات سفارش در سایت مدیریت است. این الگوی قالب مدیر admin / base_site.html سایت مدیریت Django را گسترش می دهد ، که شامل ساختار اصلی HTML و سبک های CSS سرپرست است. پرونده استاتیک سفارشی css / admin.css را با رگذاری می کنیم.

برای استفاده از پرونده های استاتیک ، باید آنها را از کدی که با این فصل آمده استفاده کنید. پرونده های استاتیک مستقر در استاتیک / فهرست برنامه سفارشات را کپی کرده و آنها را در همان مکان در پروژه خود اضافه کنید.

ما از بلوک های تعریف شده در الگوی والدین استفاده می کنیم تا محتوای خودمان را درج کنیم. ما اطلاعات مربوط به سفارش و کالاهای خریداری شده را نمایش می دهیم.

وقتی می خواهید یک الگوی سرپرستی گسترش دهید ، باید ساختار آن را بشناسید و بلوک های موجود را شناسایی کنید. می توانید تمام الگوهای سرپرست را در <https://github.com/django/django/tree/2.0/django/contrib/admin/templates/admin> پیدا کنید.

همچنین در صورت نیاز می توانید از الگوی سرپرست رد شوید. برای رد کردن یک الگوی سرپرست ، آنرا در فهرست الگوهای خود نگه دارید و همان مسیر و نام خانوادگی نسبی را نگه دارید. سایت مدیریت جنگو به جای نمونه پیش فرض از الگوی دلخواه شما استفاده خواهد کرد.

در آخر ، اجازه دهید پیوندی به هر شیء سفارش در صفحه نمایش لیست سایت مدیریت اضافه کنیم. پرونده admin.py برنامه سفارش را ویرایش کنید و کد زیر را در بالای کلاس OrderAdmin اضافه کنید:

```
from django.urls import reverse
from django.utils.safestring import mark_safe
```

```
def order_detail(obj):
    return mark_safe('<a href="{}">View</a>'.format(
        reverse('orders:admin_order_detail', args=[obj.id])))
```

این عملکردی است که یک شیء سفارش را به عنوان یک آرگومان در نظر می‌گیرد و پیوند HTML را برای آدرس اینترنتی admin_order_detail برمی‌گرداند. جنگو بطور پیش فرض از خروجی HTML فرار می‌کند. برای جلوگیری از فرار خودکار ما باید از تابع mark_safe استفاده کنیم.

سپس برای نمایش پیوند، کلاس OrderAdmin را ویرایش کنید:

```
class OrderAdmin(admin.ModelAdmin):
    list_display = ['id',
                    'first_name',
                    # ...
                    'updated',
                    order_detail]
```

/ را در مرورگر خود باز کنید. هر سطر اکنون یک پیوند مشاهده به شرح زیر دارد:

PAID	CREATED	UPDATED	ORDER DETAIL
	Feb. 6, 2018, 1:35 a.m.	Feb. 6, 2018, 1:45 a.m.	View

برای بارگیری صفحه جزئیات سفارش ، بر روی پیوند مشاهده کلیک کنید. شما باید صفحه ای مانند صفحه زیر را مشاهده کنید:

The screenshot shows the Django admin interface for an order. At the top, there's a blue header bar with the text "Django administration". Below it, a secondary header bar shows the path "Home > Orders > Order 19 > Detail". The main content area has a title "Order 19". On the right side, there's a "PRINT ORDER" button. The order details are listed in a table:

Created	Feb. 6, 2018, 1:35 a.m.
Customer	Antonio Melé
E-mail	antonio.mele@gmail.com
Address	Jazz Street, 28027 Madrid
Total amount	\$21.2
Status	Paid

Below the details, there's a section titled "Items bought" with a table:

PRODUCT	PRICE	QUANTITY	TOTAL
Tea powder	\$21.2	1	\$21.2
Total			\$21.2

تولید فاکتورهای PDF به صورت پویا

اگر چنان که سیستم پرداخت کامل و پرداخت را داریم ، می توانیم برای هر سفارش یک فاکتور PDF تهیه کنیم. چندین کتابخانه پایتون برای تولید فایل‌های PDF وجود دارد. Reportlab یکی از کتابخانه‌های محبوب برای تولید PDF با کد پایتون است. می توانید اطلاعاتی در مورد نحوه تولید فایل‌های PDF با Reportlab در اینجا بباید

<https://docs.djangoproject.com/en/2.0/howto/outputting-pdf>

در بیشتر موارد ، شما مجبور خواهید بود که سبک‌های سفارشی و قالب‌بندی را به فایل‌های PDF اضافه کنید. شما می توانید با ارائه یک قالب HTML و تبدیل آن به یک فایل PDF ، پایتون را از لایه ارائه دور نگه دارید ، راحت‌تر است. ما قصد داریم این رویکرد را دنبال کنیم و از مازول برای تولید فایل‌های PDF با Django استفاده کنیم. ما از WeasyPrint استفاده می کنیم ، که یک کتابخانه Python است که می تواند فایل‌های PDF را از قالب‌های HTML تولید کند.

نصب WeasyPrint

ابتدا وابستگی‌های WeasyPrint را برای سیستم عامل خود نصب کنید ، که در <http://weasyprint.org/docs/install/#platforms> پیدا خواهید کرد. سپس با استفاده از دستور زیر WeasyPrint را از طریق پیپ نصب کنید:

```
pip install WeasyPrint
```

ایجاد یک قالب PDF

ما به یک سند HTML به عنوان ورودی برای WeasyPrint نیاز داریم. ما قصد داریم یک الگوی HTML ایجاد کنیم ، آن را با استفاده از Django ارائه دهیم و برای تولید فایل PDF آن را به WeasyPrint منتقل کنیم.

یک فایل الگوی جدید را در برنامه سفارش ایجاد کنید و آن را pdf.html templates/orders/order/directory بنامید. کد زیر را به آن اضافه کنید:

```
<body>
  <h1>My Shop</h1>
  <p>
    Invoice no. {{ order.id }}<br>
    <span class="secondary">
      {{ order.created|date:"M d, Y" }}
    </span>
  </p>

  <h3>Bill to</h3>
  <p>
    {{ order.first_name }} {{ order.last_name }}<br>
    {{ order.email }}<br>
    {{ order.address }}<br>
    {{ order.postal_code }}, {{ order.city }}<br>
  </p>

  <h3>Items bought</h3>
  <table>
    <thead>
      <tr>
        <th>Product</th>
        <th>Price</th>
        <th>Quantity</th>
        <th>Cost</th>
      </tr>
    </thead>
    <tbody>
      {% for item in order.items.all %}<tr>
```

```

<tr class="row{% cycle "1" "2" %}">
    <td>{{ item.product.name }}</td>
    <td class="num">${{ item.price }}</td>
    <td class="num">{{ item.quantity }}</td>
    <td class="num">${{ item.get_cost }}</td>
</tr>
{% endfor %}
<tr class="total">
    <td colspan="3">Total</td>
    <td class="num">${{ order.get_total_cost }}</td>
</tr>
</tbody>
</table>

<span class="{% if order.paid %}paid{% else %}pending{% endif %}">
    {% if order.paid %}Paid{% else %}Pending payment{% endif %}
</span>
</body>
</html>

```

این الگوی فاکتور PDF است. در این الگو، تمام جزئیات سفارش و یک عنصر `<table>` از جمله محصولات را نمایش می دهیم. ما همچنین پیامی را برای نمایش اینکه آیا سفارش پرداخت شده است یا پرداخت هنوز در انتظار است، نمایش می دهیم.

PDF فایل‌های را

ما می خواهیم با استفاده از سایت مدیریت، نمایی را برای تولید فاکتورهای PDF برای سفارشات موجود ایجاد کنیم و فایل views.py را در فهرست برنامه سفارشات ویرایش کنید و کد زیر را به آن اضافه کنید:

```

from django.conf import settings
from django.http import HttpResponseRedirect
from django.template.loader import render_to_string
import weasyprint

@staff_member_required
def admin_order_pdf(request, order_id):
    order = get_object_or_404(Order, id=order_id)
    html = render_to_string('orders/order/pdf.html',
                           {'order': order})
    response = HttpResponseRedirect(content_type='application/pdf')
    response['Content-Disposition'] = 'filename=%s.pdf' % order.id
    weasyprint.HTML(string=html).write_pdf(response,
                                           stylesheets=[weasyprint.CSS(
                                               settings.STATIC_ROOT + 'css/pdf.css')])
    return response

```

این نمای تولید فاکتور PDF برای سفارش است. ما از دکوراتور `Staff_member_required` استفاده می کنیم تا اطمینان حاصل شود که فقط کاربران کارکنان می توانند به این نمای دسترسی داشته باشند. ما با شناسه داده شده دستور `Order` را می گیریم و از عملکرد `render_to_string()` ارائه شده توسط Django برای ارائه `orders/order/pdf.html` استفاده می کنیم. HTML ارائه شده در متغیر `HTML` ذخیره می شود.

سپس یک شی جدید `HttpResponse` تولید می کنیم که نوع محتوای برنامه / pdf و از جمله عنوان `Content-Dispose` را برای مشخص کردن نام پرونده مشخص می کند. ما از `WeasyPrint` برای تولید پرونده PDF از کد ارائه شده HTML استفاده می کنیم و پرونده را به شیء `HttpResponse` می نویسیم. از فایل استاتیک `css/pdf.css` استفاده کنید تا سبکهای CSS را به فایل PDF تولید شده اضافه کنید. و با استفاده از تنظیمات `STATIC_ROOT` آن را از مسیر محلی بارگیری کنید. سرانجام ، پاسخ تولید شده را بر می گردانیم.

اگر سبک های CSS را از دست داده اید ، به یاد داشته باشید که پرونده های استاتیک مستقر در `static/directory` برنامه فروشگاه را در همان محل پروژه خود کپی کنید.

از آنجا که ما باید از تنظیمات `STATIC_ROOT` استفاده کنیم ، باید آن را به پروژه خود اضافه کنیم. این مسیر پروژه برای ساکن شدن پرونده های استاتیک است. پرونده `settings.py` را ویرایش کنید و تنظیم زیر را اضافه کنید:

```
STATIC_ROOT = os.path.join(BASE_DIR, 'static/')
```

سپس دستور زیر را اجرا کنید:

```
python manage.py collectstatic
```

شما باید خروجی را مانند below مشاهده کنید

```
120 static files copied to 'code/myshop/static'.
```

فرمان استاتیک کل پرونده های استاتیک را از برنامه های شما در فهرست راهنمایی تعریف شده در تنظیمات STATIC_ROOT کپی می کند. این به هر برنامه اجازه می دهد تا فایلهای استاتیک خود را با استفاده از یک STATICFILES_DIRS موجود در آنها تهیه کند. همچنین می توانید در تنظیمات STATICFILES_DIRECTORY منابع پرونده استاتیک static/directory دیگری را ارائه دهید. تمام فهرست هایی که در لیست STATICFILES_DIRS مشخص شده اند نیز هنگام اجرای استاتیک در فهرست STATIC_ROOT کپی می شوند. هر زمان که مجدداً استاتیک را اجرا کنید ، از شما سؤال می شود که آیا می خواهید پرونده های استاتیک موجود را نادیده بگیرید.

پرونده urls.py را درون دایرکتوری برنامه سفارشات ویرایش کنید و الگوی URL زیر را به آن اضافه کنید:

```
urlpatterns = [
    # ...
    path('admin/order/<int:order_id>/pdf/',
        views.admin_order_pdf,
        name='admin_order_pdf'),
```

اکنون می توانیم صفحه نمایش لیست سروها را برای مدل Order ویرایش کنیم تا برای هر نتیجه لینک به فایل PDF اضافه کنیم. پرونده admin.py را درون برنامه سفارش ویرایش کنید و کد زیر را در بالای کلاس OrderAdmin اضافه کنید:

```
def order_pdf(obj):
    return mark_safe('<a href="{}">PDF</a>'.format(
        reverse('orders:admin_order_pdf', args=[obj.id])))
order_pdf.short_description = 'Invoice'
```

اگر یک ویژگی short_description را برای تماس گیرنده خود تعیین کنید ، Django از آن برای نام ستون استفاده می کند. را به ویژگی list_display از کلاس OrderAdmin به شرح زیر اضافه کنید:

```
class OrderAdmin(admin.ModelAdmin):
    list_display = ['id',
                    # ...
                    order_detail,
                    order_pdf]
```

/ra در مرورگر خود باز کنید. هر سطر اکنون باید پیوند PDF مانند این را شامل شود:

UPDATED

ORDER DETAIL

INVOICE

Feb. 11, 2018, 3:17 p.m. View

PDF

برای هر سفارش بر روی پی دی اف کلیک کنید. برای سفارشاتی که هنوز پرداخت نشده است ، باید یک فایل PDF تولید شده مانند پرونده زیر را ببینید:

My Shop

Invoice no. 16

Feb 01, 2018

Bill to

Antonio Mele
antonio.mele@gmail.com
Jazz Street
28033, Madrid

Items bought

Product	Price	Quantity	Cost
Green tea	\$30	1	\$30
Total	\$30		

PENDING PAYMENT

برای سفارشات پرداخت شده ، فایل PDF زیر را مشاهده خواهید کرد:

My Shop

Invoice no. 19

Feb 06, 2018

Bill to

Antonio Melé
antonio.mele@gmail.com
Jazz Street
28027, Madrid

Items bought

Product	Price	Quantity	Cost
Tea powder	\$21.2	1	\$21.2
Total			\$21.2



ارسال فایلهای PDF از طریق ایمیل

هنگامی که پرداخت موفقیت آمیز باشد ، ما یک ایمیل خودکار را از جمله فاکتور PDF تولید شده به مشتریان خود می خواهیم. پرونده views.py برنامه پرداخت را ویرایش کنید و واردات زیر را به آن اضافه کنید:

```
from django.template.loader import render_to_string
from django.core.mail import EmailMessage
from django.conf import settings
import weasyprint
from io import BytesIO
```

سپس ، در نمای Payment_process ، کد زیر را بعد از خط سفارش.save() با همان سطح تورفتگی به شرح زیر اضافه کنید:

```
def payment_process(request):
    # ...
    if request.method == 'POST':
        # ...
        if result.is_success:
            # ...
            order.save()
            # create invoice e-mail
            subject = 'My Shop - Invoice no. {}'.format(order.id)
            message = 'Please, find attached the invoice for your recent\
purchase.'
            email = EmailMessage(subject,
                                  message,
                                  'admin@myshop.com',
                                  [order.email])
            # generate PDF
            html = render_to_string('orders/order/pdf.html', {'order':
order})
            out = BytesIO()
            stylesheets=[weasyprint.CSS(settings.STATIC_ROOT +
'css/pdf.css')]
            weasyprint.HTML(string=html).write_pdf(out,
                                                    stylesheets=stylesheets)

            # attach PDF file
            email.attach('order_{}.pdf'.format(order.id),
                        out.getvalue(),
                        'application/pdf')
            # send e-mail
            email.send()

            return redirect('payment:done')
        else:
            return redirect('payment:canceled')
    else:
        # ...
```

ما از کلاس EmailMessage ارائه شده توسط Django برای ایجاد یک شی ایمیل استفاده می کنیم. سپس ، ما قالب را در متغیر HTML قرار می دهیم. فایل PDF را از الگوی ارائه شده تولید می کنیم و ما آن را به نمونه BytesIO می رسانیم ، که یک بافر بایت حافظه است. سپس ، فایل PDF تولید شده را با استفاده از روش () attach ، از جمله محتويات بافر out ، به شیء EmailMessage می چسبانیم و در آخر ، ایمیل را ارسال می کنیم.

به یاد داشته باشید که برای ارسال ایمیل ، تنظیمات SMTP خود را در پرونده settings.py پروژه تنظیم کنید. برای دیدن یک مثال کار برای پیکربندی SMTP می توانید به فصل 2 ، بهبود وبلاگ خود با ویژگی های پیشرفته مراجعه کنید. اکنون ، برای دریافت فاکتور PDF در ایمیل خود می توانید مراحل پرداخت جدید را تکمیل کنید.

خلاصه

در این فصل ، شما یک دروازه پرداخت را در پروژه خود ادغام کرده اید.

شما سایت مدیریت جنگو را سفارشی کرده اید و یاد می گیرید که چگونه فایلهاي CSV و PDF را بصورت دینامیک تولید کنید. فصل بعد بینشی در مورد بین المللی سازی و بومی سازی پروژه های جنگو به شما می دهد. همچنین شما می توانید یک سیستم کوپن ایجاد کرده و یک موتور توصیه کننده محصول بسازید.

فصل نهم

فروشگاه خود را

گسترش دهید

در فصل قبل ، شما یاد گرفتید که چگونه یک درگاه پرداخت را وارد مغازه خود کنید. همچنین یاد گرفتید که چگونه فایل‌های CSV و PDF را تولید کنید. در این فصل یک سیستم کوپن به مغازه خود اضافه می کنید. شما می آموزید که چگونه بین المللی و محلی سازی کار می کند ، و یک موتور پیشنهادی می سازید.

این فصل نکات زیر را در بر می گیرد:

ایجاد سیستم کوپن برای اعمال تخفیف

بین المللی کردن پروژه خود را اضافه کنید

استفاده از Rosetta برای مدیریت ترجمه ها

ترجمه مدل ها با استفاده از Django-Parler

ساخت موتور توصیه محصول

ایجاد سیستم کوپن

بسیاری از فروشگاه‌های آنلاین کوپن به مشتری می‌دهند که با تخفیف در خریدهای آنها قابل باخرید هستند. کوپن آنلاین معمولاً از کدی است که به کاربران داده می‌شود، برای یک بازه زمانی خاص معتبر است. کد را می‌توان یک یا چند بار باخرید کرد.

ما قصد داریم برای فروشگاه خود یک سیستم کوپن ایجاد کنیم. کوپن‌های ما برای مشتریانی که در یک بازه زمانی خاص وارد کوپن می‌شوند معتبر هستند. کوپن‌ها از نظر تعداد دفعات قابل باخرید هیچ محدودیتی نخواهند داشت و از ارزش کل سبد خرید استفاده می‌شود. برای این قابلیت، ما نیاز به ایجاد یک مدل برای ذخیره کد کوپن، یک بازه زمانی معتبر و تخفیف برای اعمال آن داریم.

با استفاده از دستور زیر یک برنامه جدید را در داخل پروژه myshop ایجاد کنید:

```
python manage.py startapp coupons
```

پرونده تنظیمات mypy را ویرایش کنید و برنامه را به صورت زیر به تنظیمات INSTALLED_APPS اضافه کنید:

```
INSTALLED_APPS = [
    # ...
    'coupons.apps.CouponsConfig',
]
```

برنامه جدید اکنون در پروژه Django ما فعال است.

ساخت مدل های کوپن

بیایید با ایجاد مدل کوپن شروع کنیم. پرونده model.py را ویرایش کرده و کد زیر را به آن اضافه کنید:

```
from django.db import models
from django.core.validators import MinValueValidator,
                                  MaxValueValidator

class Coupon(models.Model):
    code = models.CharField(max_length=50,
                           unique=True)
    valid_from = models.DateTimeField()
    valid_to = models.DateTimeField()
    discount = models.IntegerField(
        validators=[MinValueValidator(0),
                   MaxValueValidator(100)])
    active = models.BooleanField()

    def __str__(self):
        return self.code
```

این مدلی است که قصد داریم از آن برای ذخیره کوپن‌ها استفاده کنیم. مدل کوپن شامل قسمتهای زیر است:

:code کدی که کاربران برای اعمال کوپن در خرید خود باید وارد کنند.

:valid_from مقدار DateTime که زمان اعتبار کوپن را نشان می‌دهد.

:valid_to مقدار DateTime که نشان می‌دهد بی‌اعتبار بودن کوپن است.

:discount نرخ تخفیف برای اعمال (این یک درصد است، بنابراین مقادیر را از 0 تا 100 می‌گیرد). ما از اعتبارگرها برای این زمینه استفاده می‌کنیم تا حداقل و حداً کثیر مقادیر پذیرفته شده را محدود کنیم.

:active یک بولی که نشان می‌دهد کوپن فعال است یا خیر.

دستور العمل زیر را برای تولید مهاجرت اولیه برای استفاده از کوپن اجرا کنید:

```
python manage.py makemigrations
```

خروجی باید شامل خطوط زیر باشد:

```
Migrations for 'coupons':  
  coupons/migrations/0001_initial.py:  
    - Create model Coupon
```

سپس ، دستور بعدی را برای اعمال مهاجرت اجرا می کنیم:

```
python manage.py migrate
```

باید خروجی را مشاهده کنید که شامل خط زیر است:

```
Applying coupons.0001_initial... OK
```

مهاجرت اکنون در پایگاه داده اعمال می شود. باید مدل کوپن را به سایت مدیریت اضافه کنیم. پرونده admin.py کوپن را ویرایش کرده و کد زیر را به آن اضافه کنید:

```
from django.contrib import admin
from .models import Coupon

class CouponAdmin(admin.ModelAdmin):
    list_display = ['code', 'valid_from', 'valid_to',
                    'discount', 'active']
    list_filter = ['active', 'valid_from', 'valid_to']
    search_fields = ['code']
```

```
admin.site.register(Coupon, CouponAdmin)
```

مدل کوپن اکنون در سایت مدیریت ثبت شده است. اطمینان حاصل کنید که سرور محلی شما با دستور اجرا کننده `python management.py runserver` اجرا شده است. می‌توانید آن را در آدرس `http://127.0.0.1:8000/admin/coupons/coupon/add` مشاهده کنید: مرورگر خود باز کنید. فرم زیر را باید مشاهده کنید:

Django administration

WELCOME ADMIN / VIEW SITE / CHANGE PASSWORD / LOG OUT

Home > Coupons > Coupons > Add coupon

Add coupon

Code:

Valid from:

Date: Today 

Time: Now 

Note: You are 1 hour ahead of server time.

Valid to:

Date: Today 

Time: Now 

Note: You are 1 hour ahead of server time.

Discount:

Active

برای ایجاد یک کوپن جدید که برای تاریخ جاری معتبر است ، فرم را پر کنید و مطمئن شوید که کادر فعل Active را بررسی کرده و روی دکمه SAVE کلیک کنید.

اعمال کوپن به سبد خرید

ما می توانیم کوپن های جدیدی را ذخیره کنیم و برای بازیابی کوپن های موجود ، نمایش داده شد. حال ما به روشنی نیاز داریم تا مشتریان بتوانند کوپن های خرید خود را اعمال کنند. قابلیت استفاده از کوپن به شرح زیر است:

1. کاربر محصولات را به سبد خرید اضافه می کند.
2. کاربر می تواند یک کد کوپن را به شکلی که در صفحه جزئیات سبد خرید نمایش داده شده است وارد کند.
3. هنگامی که یک کاربر وارد یک کد کوپن شده و فرم را ارسال می کند ، ما به دنبال یک کوپن موجود با کد مشخص شده هستیم که در حال حاضر معتبر است. ما باید بررسی کنیم که کد کوپن مطابق با کد وارد شده توسط کاربر است که ویژگی عملکرد صحیح است ، و اینکه `DateTime` فعلی بین مقادیر `معتبر_to` و `معتبر_from` است.
- 4- در صورت یافتن کوپن ، آن را در جلسه کاربر ذخیره می کنیم و سبد خرید را نشان می دهیم ، از جمله تخفیف اعمال شده بر روی آن و مبلغ کل به روز شده.
- 5- وقتی کاربر سفارش می دهد ، ما کوپن را به سفارش داده شده ذخیره می کنیم.

یک پرونده جدید را در فهرست برنامه کوپن ها ایجاد کرده و آنرا `form.py` بنامید. کد زیر را به آن اضافه کنید:

```
from django import forms
```

```
class CouponApplyForm(forms.Form):
    code = forms.CharField()
```

این روشی است که ما می خواهیم برای وارد کردن کد کوپن از کاربر استفاده کنیم. پرونده views.py را درون برنامه کوپن ویرایش کرده و کد زیر را به آن اضافه کنید:

```

from django.shortcuts import render, redirect
from django.utils import timezone
from django.views.decorators.http import require_POST
from .models import Coupon
from .forms import CouponApplyForm

@require_POST
def coupon_apply(request):
    now = timezone.now()
    form = CouponApplyForm(request.POST)
    if form.is_valid():
        code = form.cleaned_data['code']
        try:
            coupon = Coupon.objects.get(code__iexact=code,
                                         valid_from__lte=now,
                                         valid_to__gte=now,
                                         active=True)
            request.session['coupon_id'] = coupon.id
        except Coupon.DoesNotExist:
            request.session['coupon_id'] = None
    return redirect('cart:cart_detail')

```

نمای کوپن_کوپن ، کوپن را تأیید می کند و آن را در جلسه کاربر ذخیره می کند. ما برای این محدودیت در درخواست های POST از دکوراتور requ_POST استفاده می کنیم. در نمای ، ما کارهای زیر را انجام می دهیم:

1. فرم CouponApplyForm را با استفاده از داده های ارسال شده فوری می کنیم و بررسی می کنیم که فرم معتبر است.
- 2- در صورت صحت فرم ، کدی را که از سوی کاربر وارد شده است ، از فرهنگ لغت cleaned_data فرم می گیریم. ما سعی می کنیم با کد مشخص شده شیء کوپن را بازیابی کنیم. ما از جستجوی درست فیلد برای انجام یک مسابقه دقیق غیر حساس استفاده می کنیم. کوپن باید در حال حاضر فعال (active=True) باشد و برای DateTime فعلی معتبر باشد. ما از تابع timezone.now () Django برای بدست آوردن منطقه DateTime آگاه از منطقه زمانی فعلی استفاده می کنیم و آن را با زمینه های معتبر from و valid_to مقایسه می کنیم که عملکردهای let (کمتر از یا مساوی با) را انجام می دهیم و به ترتیب (بیشتر از یا مساوی با) جستجو می کنیم. .

3. ما شناسه کوپن را در جلسه کاربر ذخیره می کنیم.

4. ما کاربر را به آدرس اینترنتی cart_detail هدایت می کنیم تا سبد خرید را با کوپن اعمال شده نشان دهد.

برای مشاهده کوپن_apply به الگوی URL نیاز دارید. یک پرونده جدید را در فهرست برنامه کوپن ایجاد کرده و نام آن را urls.py بگذارید. کد زیر را به آن اضافه کنید:

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

app_name = 'coupons'

urlpatterns = [
    path('apply/', views.coupon_apply, name='apply'),
]
```

سپس urls.py اصلی پروژه myshop را ویرایش کنید و الگوهای URL کوپن ها را به شرح زیر وارد کنید:

```
urlpatterns = [
    # ...
    path('coupons/', include('coupons.urls', namespace='coupons')),
    path('', include('shop.urls', namespace='shop')),
]
```

به یاد داشته باشید که این الگوی را قبل از الگوی shop.urls قرار دهید.

اکنون پرونده cart.py برنامه سبد خرید را ویرایش کنید. واردات زیر را درج کنید:

```
from coupons.models import Coupon
```

برای شروع اولیه کوپن از جلسه فعلی ، کد زیر را به انتهای روش __init__ کلاس Cart اضافه کنید:

```
class Cart(object):
    def __init__(self, request):
        # ...
        # store current applied coupon
        self.coupon_id = self.session.get('coupon_id')
```

در این کد ، ماسعی می کنیم که کلید جلسه coupon_id را از جلسه فعلی بگیریم و مقدار آن را در شی Cart ذخیره کنیم.
روش های زیر را به موضوع Cart اضافه کنید:

```
class Cart(object):
    # ...
    @property
    def coupon(self):
        if self.coupon_id:
            return Coupon.objects.get(id=self.coupon_id)
        return None

    def get_discount(self):
        if self.coupon:
            return (self.coupon.discount / Decimal('100')) \
                * self.get_total_price()
        return Decimal('0')

    def get_total_price_after_discount(self):
        return self.get_total_price() - self.get_discount()
```

این روش ها به شرح زیر است:

کوپن () : ما این روش را به عنوان ویژگی تعریف می کنیم. اگر سبد خرید دارای یک ویژگی coupon_id باشد ، شیء کوپن با شناسه مشخص بازگردانده می شود.

کسر می شود ، باز می گردیم. get_discount () : اگر سبد خرید شامل یک کوپن است ، ما نرخ تخفیف آن را بازیابی می کنیم و مبلغ را که از کل مبلغ آن

() get_discount() : ما مبلغ کل سبد خرید را پس از کسر مبلغ برگشتی با روش get_total_price_after_discount کسر می کنیم.

کلاس Cart اکنون آماده رسیدگی به کوپن اعمال شده در جلسه فعلی و اعمال تخفیف مربوطه است. باید سیستم کوپن را در نمای جزئیات سبد خرید وارد کنیم. پرونده views.py برنامه سبد خرید را ویرایش کنید و واردات زیر را در بالای پرونده اضافه کنید:

```
from coupons.forms import CouponApplyForm
```

در پایین پایین ، نماد cart_detail را ویرایش کنید و فرم جدید را به شرح زیر اضافه کنید:

```
def cart_detail(request):
    cart = Cart(request)
    for item in cart:
        item['update_quantity_form'] = CartAddProductForm(
            initial={'quantity': item['quantity'],
                      'update': True})
    coupon_apply_form = CouponApplyForm()

    return render(request,
                  'cart/detail.html',
                  {'cart': cart,
                   'coupon_apply_form': coupon_apply_form})
```

قالب سبد / detail.html برنامه سبد خرید را ویرایش کرده و خطوط زیر را پیدا کنید:

```
<tr class="total">
  <td>Total</td>
  <td colspan="4"></td>
  <td class="num">${{ cart.get_total_price }}</td>
```

آنها را با موارد زیر جایگزین کنید:

```
{% if cart.coupon %}
  <tr class="subtotal">
    <td>Subtotal</td>
    <td colspan="4"></td>
    <td class="num">${{ cart.get_total_price|floatformat:"2" }}</td>
  </tr>
  <tr>
    <td>
      "{{ cart.coupon.code }}" coupon
      ({{ cart.coupon.discount }}% off)
    </td>
    <td colspan="4"></td>
    <td class="num neg">
      - ${{ cart.get_discount|floatformat:"2" }}
    </td>
  </tr>
{% endif %}
<tr class="total">
  <td>Total</td>
  <td colspan="4"></td>
  <td class="num">
    ${{ cart.get_total_price_after_discount|floatformat:"2" }}
  </td>
</tr>
```

این کد برای نمایش یک کوپن اختیاری و نرخ تخفیف آن است. اگر سبد خرید شامل یک کوپن باشد، ما یک ردیف اول، از جمله مقدار کل سبد را به عنوان زیرمجموعه نمایش می‌دهیم. سپس از یک ردیف دوم برای نمایش کوپن فعلی اعمال شده روی سبد استفاده می‌کنیم. سرانجام، با فراخوانی متدهای `get_total_price_after_discount()` شیء سبد خرید، قیمت کلی را شامل می‌شود.

در همان پرونده ، کد زیر را بعد از برجسب <table> HTML وارد کنید:

```
<p>Apply a coupon:</p>
<form action="{% url "coupons:apply" %}" method="post">
    {{ coupon_apply_form }}
    <input type="submit" value="Apply">
```

```
    {% csrf_token %}
</form>
```

با استفاده از این فرم می توانید کد کوپن را وارد کرده و آن را در سبد خرید فعلی اعمال کنید.

http://127.0.0.1:8000 / را در مرورگر خود باز کنید ، محصولی را به سبد خرید اضافه کنید ، و با وارد کردن کد آن در فرم ، کوپن ایجاد شده را اعمال کنید.

باید دید که سبد خرید تخفیف کوپن را به شرح زیر نشان می دهد:

Your shopping cart

Image	Product	Quantity	Remove	Unit price	Price
	Tea powder	1 <input type="button" value="Update"/>	Remove	\$21.2	\$21.2
Subtotal					\$21.20
'SUMMER' coupon (10% off)					-\$2.12
Total					\$19.08

Apply a coupon:

Code:

باید کوپن را به مرحله بعدی فرایند خرید اضافه کنیم. قالب orders/order/create.html برنامه سفارشات را ویرایش کنید و خطوط زیر را پیدا کنید:

```
<ul>
  {% for item in cart %}
    <li>
      {{ item.quantity }}x {{ item.product.name }}
      <span>${{ item.total_price }}</span>
    </li>
```

```
  {% endfor %}
</ul>
```

آنها را با کد زیر جایگزین کنید:

```
<ul>
  {% for item in cart %}
    <li>
      {{ item.quantity }}x {{ item.product.name }}
      <span>${{ item.total_price|floatformat:"2" }}</span>
    </li>
  {% endfor %}
  {% if cart.coupon %}
    <li>
      "{{ cart.coupon.code }}" ({{ cart.coupon.discount }}% off)
      <span>- ${{ cart.get_discount|floatformat:"2" }}</span>
    </li>
  {% endif %}
</ul>
```

در صورت وجود خلاصه سفارش باید کوپن اعمال شده را نیز شامل شود. اکنون خط زیر را پیدا کنید:

```
<p>Total: ${{ cart.get_total_price }}</p>
```

آن را با موارد زیر جایگزین کنید:

```
<p>Total: ${{ cart.get_total_price_after_discount|floatformat:"2" }}</p>
```

با این کار ، با اعمال تخفیف کوپن ، قیمت کل نیز محاسبه می شود.

در مرورگر خود <http://127.0.0.1:8000/order/create> را باز کنید. باید ببینید که خلاصه سفارش شامل کوپن کاربردی

به شرح زیر است:

Your order

- 1x Tea powder \$21.20
- "SUMMER" (10% off) - \$2.12

Total: \$19.08

کاربران هم اکنون می توانند کوپن ها را به سبد خرید خود اعمال کنند. با این حال ، ما هنوز نیاز داریم که اطلاعات کوپن را به ترتیبی که ایجاد می شود در هنگام چک کردن سبد خرید ، ذخیره کنیم.

اعمال کوپن به سفارشات

ما قصد داریم کوپنی را که برای هر سفارش اعمال شده بود ، ذخیره کنیم.

در ابتدا ، اگر وجود داشته باشد ، باید مدل Order را برای ذخیره شیء مربوط به کوپن اصلاح کنیم.

پرونده model.py برنامه سفارش را ویرایش کنید و واردات زیر را به آن اضافه کنید:

```
from decimal import Decimal
from django.core.validators import MinValueValidator, \
                                  MaxValueValidator
from coupons.models import Coupon
```

سپس قسمت های زیر را به مدل Order اضافه کنید:

```
class Order(models.Model):
    # ...
    coupon = models.ForeignKey(Coupon,
                               related_name='orders',
                               null=True,
                               blank=True,
                               on_delete=models.SET_NULL)
    discount = models.IntegerField(default=0,
                                   validators=[MinValueValidator(0),
                                   MaxValueValidator(100)])
```

این زمینه ها به ما امکان می دهند یک کوپن اختیاری را برای سفارش و درصد تخفیف اعمال شده با کوپن ذخیره کنیم. تخفیف در شیء مربوط به کوپن ذخیره می شود، اما ما آن را در مدل Order درج می کنیم تا در صورت اصلاح یا حذف کوپن، آن را حفظ کنیم. ما on_delete model.SET_NULL را بر روی قرار می دهیم تا در صورت حذف کوپن، قسمت کوپن روی Null تنظیم شود.

ما باید یک مهاجرت ایجاد کنیم تا زمینه های جدید مدل Order را در بر گیریم. دستور زیر را از خط فرمان اجرا کنید:

```
python manage.py makemigrations
```

باید خروجی مانند موارد زیر را مشاهده کنید:

```
Migrations for 'orders':
  orders/migrations/0003_auto_20180307_2202.py:
    - Add field coupon to order
    - Add field discount to order
```

مهاجرت جدید را با دستور زیر اعمال کنید:

```
python manage.py migrate orders
```

باید تأییدی را ببینید که نشان می دهد مهاجرت جدید اعمال شده است. تغییرات فیلد سفارش اکنون با بانک اطلاعات همگام شده است.

به پرونده model.py برگردید و روش get_total_cost() مدل Order را به شرح زیر تغییر دهید:

```
class Order(models.Model):
    # ...
    def get_total_cost(self):
        total_cost = sum(item.get_cost() for item in self.items.all())
        return total_cost - total_cost * \
            (self.discount / Decimal('100'))
```

روش get_total_cost() از مدل Order اکنون تخفیف اعمال شده را در صورت وجود ، در نظر می گیرد.

پرونده views.py را از برنامه سفارش ویرایش کنید و جهت صرفه جویی در کوپن مربوطه و تخفیف آن هنگام ایجاد سفارش جدید ، ترتیب rend_create را تغییر دهید. خط زیر را پیدا کنید:

```
order = form.save()
```

آن را با موارد زیر جایگزین کنید:

```
order = form.save(commit=False)
if cart.coupon:
    order.coupon = cart.coupon
    order.discount = cart.coupon.discount
order.save()
```

در کد جدید ، با استفاده از روش `save()` فرم `OrdercreateForm` یک شیء سفارش ایجاد می کنیم. ما هنوز از ذخیره کردن آن در دیتابیس با استفاده از `commit = False` پرهیز می کنیم. اگر سبد خرید شامل یک کوپن باشد ، ما کوپن مربوطه و تخفیف اعمال شده را ذخیره می کنیم. سپس شیء سفارش را در پایگاه داده ذخیره می کنیم.

اطمینان حاصل کنید که سرور توسعه با دستور `python management.py runserver` در حال اجرا است.

`/http://127.0.0.1:8000` را در مرورگر خود باز کرده و با استفاده از کوپن ایجاد شده ، خرید را انجام دهید. با خرید موفق ، می توانید به `/http://127.0.0.1:8000/admin/order/order` بروید و بررسی کنید که شیء سفارش شامل کوپن و تخفیف کاربردی به شرح زیر است:

Brantree id: d31natz6

Coupon: SUMMER ✓ ✕ ✖

Discount: 10

ORDER ITEMS			
PRODUCT	PRICE	QUANTITY	DELETE?
21			
3 Q Tea powder	21,2	1	

همچنین می توانید الگوی جزئیات سفارش سفارش مدیر و قبض PDF سفارش را تغییر دهید تا کوپن کاربردی به همان روشی که برای سبد خرید انجام دادیم نمایش داده شود.

در مرحله بعد ، ما می خواهیم بین المللی سازی را به پروژه خود بیفزاییم.

افزودن بین المللی و بومی سازی

جنگو پشتیبانی بین المللی و محلی سازی کاملی را ارائه می دهد. این برنامه به شما امکان می دهد تا برنامه خود را به چندین زبان ترجمه کنید و از قالب بندی های محلی برای تاریخ ها ، زمان ها ، اعداد و مناطق زمانی استفاده می کند. بیایید تفاوت بین بین المللی و بومی سازی را روشن کنیم. بین المللی سازی (اغلب به اختصار n18) فرایند سازگاری نرم افزار برای استفاده بالقوه از زیان ها و محلی های مختلف است ، به طوری که برای یک زیان یا محلی خاص سخت نیست. محلی سازی (به اختصار n10) فرایندی است که در واقع ترجمه نرم افزار و تطبیق آن با محلی خاص انجام می شود. خود جنگو با استفاده از چارچوب بین المللی سازی خود به بیش از 50 زیان ترجمه شده است.

بین المللی شدن با جنگو

چارچوب بین المللی سازی به شما امکان می دهد رشته ها را برای ترجمه به راحتی در کد پایتون و در قالب های خود علامت گذاری کنید. این نرم افزار برای تولید و مدیریت پرونده های پیام به ابزار ابزار gettext گنو متکی است. فایل پیام یک فایل متغیر ساده است که یک زبان را نشان می دهد.

این شامل بخشی یا همه رشته های ترجمه موجود در برنامه شما و ترجمه های مربوطه آنها برای یک زبان واحد است. پرونده های پیام دارای پسوند po هستند.

پس از انجام ترجمه ، فایلهای پیام برای دسترسی سریع به رشته های ترجمه شده کامپایل می شوند. فایلهای ترجمه کامپایل شده دارای پسوند mo هستند.

تنظیمات بین المللی سازی و بومی سازی

Django تنظیمات مختلفی را برای بین المللی شدن ارائه می دهد. تنظیمات زیر مهمترین موارد زیر است:

USE_I18N: بولی که مشخص می کند سیستم ترجمه Django فعال شده است یا خیر. این به طور پیش فرض صحیح است.

USE_L10N: بولی که نشان می دهد قالب بندی محلی فعال است یا خیر. در صورت فعال ، از قالب های محلی برای نشان دادن تاریخ و شماره استفاده می شود. این به طور پیش فرض نادرست است.

USE_TZ: بولی که مشخص می کند آیا تاریخ ها از زمان منطقه آگاه هستند یا خیر. وقتی پروژه ای را با دستور star star star TZ ایجاد می کنید ، این گزینه روی True قرار دارد.

LANGUAGE_CODE: کد زبان پیش فرض برای پروژه. به عنوان مثال ، در قالب استاندارد ID زبان ، 'en-us' برای انگلیسی آمریکایی یا 'en-GB' برای انگلیسی انگلیسی است. برای این کار باید تنظیم شود USE_I18N روی صحبت تنظیم شود تا عملی شود. لیستی از شناسه های معتبر زبان را می توانید در اینجا پیدا کنید

<http://www.i18nguy.com/unicode/language-identifiers.html>

زیانها: طاقچه ای که شامل زیان های در دسترس برای این پروژه است. آنها در دو نوبت از کد زیان و نام زیان آمده اند. لیست زیان های موجود را می توانید در `django.conf.global_settings` مشاهده کنید. وقتی انتخاب می کنید که در چه زبانی سایت شما در دسترس باشد ، `LANGUAGES` را به زیر مجموعه آن لیست تنظیم می کنید.

`LOCALE_PATHS`: لیستی از دایرکتوری هایی که Django به دنبال پرونده های پیام است که حاوی ترجمه برای این پروژه است.

`TIME_ZONE`: رشته ای که منطقه زمانی پروژه را نشان می دهد. هنگامی که یک پروژه جدید را با استفاده از دستور شروع پروژه شروع می کنید ، روی "UTC" تنظیم می شود. می توانید آن را در هر منطقه زمانی دیگر ، مانند 'اروپا / مادرید' تنظیم کنید.

دستورات مدیریت بین المللی

جنگو شامل دستورات مدیریتی زیر برای مدیریت ترجمه ها می باشد:

ایجاد پیام: این کار روی درخت مبدأ اجرا می شود تا تمام رشته های مشخص شده برای ترجمه را پیدا کند و پرونده های پیام po. را در فهرست محلی ایجاد یا به روز کند. یک پرونده po برای هر زبان ایجاد می شود.

کامپایل پیام ها: این پرونده های پیغام po موجود را به پرونده mo که برای بازیابی ترجمه ها استفاده می شود ، کامپایل می کند.

برای ایجاد ، به روزرسانی و کامپایل فایل های پیام به مجموعه ابزار `gettext` احتیاج دارید. بیشتر توزیعهای لینوکس شامل ابزار `gettext` است. اگر از macOS X استفاده می کنید ، احتمالاً ساده ترین راه نصب آن از طریق Homebrew در /`brew install gettext` است. همچنین ممکن است لازم باشد که آن را به زور پیوند دهید. برای ویندوز ، مراحل <https://docs.djangoproject.com/en/2.0/topics/i18n/translation/#gettext-on-windows> را دنبال کنید.

چگونه ترجمه ها را به یک پروژه Django اضافه کنیم

بیایید نگاهی به روند انجام بین المللی پروژه خود بیندازیم. ما باید موارد زیر را انجام دهیم:

1. رشته ها را برای ترجمه در کد پایتون و الگوهای ما علامت گذاری کنید
2. دستور ایجاد پیام ها را ایجاد کنید تا پرونده های پیام که شامل کلیه رشته های ترجمه از کد ما هستند ایجاد یا به روز کنید
3. رشته های موجود در پرونده های پیام را ترجمه کرده و آنها را با استفاده از دستور مدیریت پیام کامپایل کنید

چگونه جنگو زبان فعلی را تعیین می کند

جنگو با یک واسطه ارائه می شود که زبان فعلی را بر اساس داده های درخواستی تعیین می کند. این میان افزار LocaleMiddleware است که در django.middleware.locale.LocaleMiddleware ساکن است.

کارهای زیر را انجام می دهد:

1. اگر از `patterns` استفاده می کنید ، یعنی از الگوهای URL ترجمه شده استفاده می کنید ، به دنبال یک پیشوند زبان در URL درخواستی است تا زبان فعلی را تعیین کند.
2. اگر پیشوند زبانی یافت نشد ، به دنبال `LANGUAGE_SESSION_KEY` موجود در جلسه کاربر فعلی است.
3. اگر زبان در جلسه تنظیم نشده است ، به دنبال یک کوکی موجود با زبان فعلی است. نام سفارشی برای این کوکی را می توان در تنظیمات `LANGUAGE_COOKIE_NAME` ارائه داد. به طور پیش فرض ، نام این کوکی `django_language` است.
- 4- اگر هیچ کوکی پیدا نشد ، به دنبال سرصفحه `Accept-Language` HTTP درخواست می باشد.
- 5- اگر هدر `Accept-Language` زبانی را مشخص نمی کند ، Django از زبانی تعریف شده در تنظیمات `LANGUAGE_CODE` استفاده می کند. به طور پیش فرض ، جنگو از زبان تعریف شده در تنظیمات `LANGUAGE_CODE` استفاده می کند مگر اینکه از LocaleMiddleware استفاده کنید. فرآیند شرح داده شده در اینجا فقط هنگام استفاده از این واسطه کاربرد دارد.

آماده سازی پروژه ما برای بین المللی شدن

بیایید پروژه خود را برای استفاده از زیان‌های مختلف آماده کنیم. ما قصد داریم یک نسخه انگلیسی و اسپانیایی برای فروشگاه خود بسازیم. پرونده settings.py پروژه خود را ویرایش کنید و تنظیمات زیر را در زیر تنظیمات آن را در کنار تنظیمات قرار دهید:

```
LANGUAGES = (  
    ('en', 'English'),  
    ('es', 'Spanish'),  
)
```

تنظیمات LANGUAGES شامل دو تاپ است که از یک کد زیان و یک نام تشکیل شده است. کدهای زیان می‌توانند مخصوص محلی باشند، مانند منوها یا en-gb یا عمومی مانند en. با این تنظیم، ما مشخص می‌کنیم که برنامه ما فقط به زیان انگلیسی و اسپانیایی در دسترس خواهد بود. اگر تنظیمات LANGUAGES سفارشی را تعریف نکنیم، سایت به تمام زبانهایی که Django به آنها ترجمه شده است، در دسترس خواهد بود.

تنظیمات LANGUAGE_CODE خود را به شرح زیر بنویسید:

```
LANGUAGE_CODE = 'en'
```

اطمینان حاصل کنید که این میان افزار پس از SessionMiddleware آمده است زیرا LocaleMiddleware باید از داده های جلسه استفاده کند. همچنین باید قبل از CommonMiddleware قرار داده شود زیرا دومی برای حل URL درخواست شده به یک زیان فعال نیاز دارد. اکنون تنظیمات MiddleWARE باید به شرح زیر باشد:

```
MIDDLEWARE = [
```

```
    'django.middleware.security.SecurityMiddleware',
    'django.contrib.sessions.middleware.SessionMiddleware',
    'django.middleware.locale.LocaleMiddleware',
    'django.middleware.common.CommonMiddleware',
    # ...
]
```

ساختار دایرکتوری زیر را در فهرست اصلی پروژه ، در کنار پرونده management.py ایجاد کنید:

```
locale/
    en/
    es/
```

فهرست محلی ، محلی است که پرونده های پیام برای برنامه شما اقامت دارد. دوباره پرونده تنظیمات را ویرایش کنید و تنظیمات زیر را به آن اضافه کنید:

```
LOCALE_PATHS = (
    os.path.join(BASE_DIR, 'locale/'),
)
```

تنظیم LOCALE_PATHS دایرکتوری های را که Django باید برای جستجوی فایل های ترجمه جستجو کند را مشخص می کند. مسیرهای محلی که برای اولین بار ظاهر می شوند دارای بالاترین تقدیم.

هنگامی که از دستور make messages از فهرست پروژه خود استفاده می کنید ، پرونده های پیام در محلی / مسیری که ایجاد کردیم تولید می شوند. با این حال ، برای برنامه های حاوی محلی / فهرست ، پرونده های پیام در آن فهرست ایجاد می شوند.

ترجمه کد پایتون

برای ترجمه ادبیات در کد پایتون ، می توانید رشته هایی را برای ترجمه با استفاده از عملکرد gettext() موجود در django.utils.translation علامت گذاری کنید. این عملکرد پیام را ترجمه کرده و یک رشته را برعی گرداند. این کنوانسیون قرار است این تابع را به عنوان نام مستعار کوتاه تر به named underscore character _ (underscore character) معرفی کند.

می توانید تمام اسناد مربوط به ترجمه ها را در اینجا بباید

<https://docs.djangoproject.com/fa/2.0/topics/i18n/translation>

ترجمه های استاندارد

کد زیر نحوه نشان دادن رشته ای برای ترجمه را نشان می دهد:

```
from django.utils.translation import gettext as _
output = _('Text to be translated.')
```

ترجمه های تنبیل

جنگو برای تمام عملکردهای ترجمه خود دارای نسخه های تنبیل است که پسوند lazy () دارند. هنگام استفاده از توابع تنبیل ، رشته ها هنگام دستیابی به مقدار ترجمه می شوند ، نه اینکه وقتی تابع فراخوانی می شود (به همین دلیل آنها ترجمه شده اند)

با تنبیل). توابع ترجمه تنبیل وقتی به کار می روند که رشته های مشخص شده برای ترجمه در مسیری باشند که هنگام بارگذاری ماژول ها اجرا می شوند.

ترجمه از جمله متغیرها

رشته های مشخص شده برای ترجمه می توانند شامل متغیرهایی در ترجمه شوند. کد زیر نمونه ای از رشته ترجمه با نگهدارنده محل است:

```
from django.utils.translation import gettext as _
month = _('April')
day = '14'
output = _('Today is %(month)s %(day)s') % {'month': month,
                                              'day': day}
```

با استفاده از متغیرهای می توانید متغیرهای متن را دوباره ترتیب دهید. به عنوان مثال ، ترجمه انگلیسی به عنوان قبلی ممکن است "امروز 14 آوریل" باشد ، در حالی که اسپانیایی "Hoy es 14 de Abril" است.

هنگامی که بیش از یک پارامتر برای رشته ترجمه دارد ، همیشه از جایگذاری رشته استفاده کنید.

با انجام این کار ، شما می توانید متن حفره یا سوراخ را دوباره ترتیب دهید.

اشکال جمع در ترجمه

برای اشکال جمع ، می توانید از `ngettext()` و `ngettext_lazy()` استفاده کنید. این توابع بسته به یک استدلال که تعداد اشیاء را نشان می دهد ، اشکال مفرد و جمع را ترجمه می کنند. مثال زیر نحوه استفاده از آنها را نشان می دهد:

```
output = ngettext('there is %(count)d product',
                  'there are %(count)d products',
                  count) % {'count': count}
```

اکنون که اصول ترجمه ادبیات در کد پایتون را می دانید ، وقت آن رسیده است که ترجمه های خود را برای پروژه ما اعمال کنید.

ترجمه کد خود را

پرونده settings.py پروژه خود را ویرایش کنید ، عملکرد gettext_lazy() را وارد کنید و تنظیمات LANGUAGES را به شرح زیر تغییر دهید تا نام های زبان را ترجمه کنید:

```
from django.utils.translation import gettext_lazy as _

LANGUAGES = (
    ('en', _('English')),
    ('es', _('Spanish')),
)
```

در اینجا ، ما از تابع gettext_lazy() به جای () برای جلوگیری از واردات دایرہ ای استفاده می کنیم ، بنابراین نامهای زبان ها را هنگام دسترسی ترجمه می کنیم.

پوسته را باز کرده و دستور زیر را از فهرست پروژه خود اجرا کنید:

```
django-admin makemessages --all
```

باید خروجی زیر را مشاهده کنید:

```
processing locale es  
processing locale en
```

نگاهی به .locale/ directory باید یک ساختار فایل مانند زیر را ببینید:

```
en/  
    LC_MESSAGES/  
        django.po
```

```
es/  
    LC_MESSAGES/  
        django.po
```

یک فایل پیغام .po برای هر زبان ایجاد شده است. es / LC_MESSAGES / django.po را با یک ویرایشگر متن باز کنید.
در انتهای پرونده باید موارد زیر را مشاهده کنید:

```
#: myshop/settings.py:117
msgid "English"
msgstr ""

#: myshop/settings.py:118
msgid "Spanish"
msgstr ""
```

قبل از هر رشته ترجمه ، توضیحی در مورد پرونده و خط مورد نظر درج شده است. هر ترجمه شامل دو رشته است:

msgid: رشته ترجمه همانطور که در کد منبع ظاهر می شود.

msgstr: ترجمه زیان که بصورت پیش فرض خالی است.

اینجاست که باید ترجمه واقعی را برای رشته داده شده وارد کنید.

ترجمه های msgstr را برای رشته msg داده شده به شرح زیر پر کنید:

```
#: myshop/settings.py:117
msgid "English"
msgstr "Inglés"

#: myshop/settings.py:118
msgid "Spanish"
msgstr "Español"
```

فایل پیام تغییر یافته را ذخیره کنید ، پوسته را باز کنید و دستور زیر را اجرا کنید:

```
django-admin compilemessages
```

اگر همه چیز خوب پیش رفت ، باید خروجی مانند موارد زیر را ببینید:

```
processing file django.po in myshop/locale/en/LC_MESSAGES
processing file django.po in myshop/locale/es/LC_MESSAGES
```

خروجی اطلاعاتی راجع به پرونده های پیغام تهیه شده به شما می دهد. دوباره نگاهی به فهرست محلی پروژه Myshop بیندازید. باید پرونده های زیر را مشاهده کنید:

```
en/
  LC_MESSAGES/
    django.mo
    django.po
es/
  LC_MESSAGES/
    django.mo
    django.po
```

مشاهده می کنید که یک فایل پیغام کامپایل شده mo برای هر زبان تولید شده است.

ما خود اسمی زیانها را ترجمه کرده ایم. حال باید نام فیلد های مدل را که در سایت نمایش داده می شود ترجمه کنیم. پرونده برنامه سفارش را ویرایش کنید و نام های مشخص شده برای ترجمه را برای قسمت های مدل Order به صورت زیر اضافه کنید:

```
from django.utils.translation import gettext_lazy as _

class Order(models.Model):
    first_name = models.CharField(_('first name'),
                                  max_length=50)
    last_name = models.CharField(_('last name'),
                                 max_length=50)
    email = models.EmailField(_('e-mail'))
    address = models.CharField(_('address'),
                               max_length=250)
    postal_code = models.CharField(_('postal code'),
                                   max_length=20)
```

```
    city = models.CharField(_('city'),
                           max_length=100)
    # ...
```

ما نامهایی را برای قسمتهایی که هنگام تنظیم سفارش جدید کاربر نمایش داده می شود اضافه کردیم. اینها `firstname_name` ، نام خانوادگی ، ایمیل ، آدرس ، کد پستی و شهر هستند. به یاد داشته باشید که می توانید از ویژگی `verbose_name` نیز برای نامگذاری فیلدها استفاده کنید.

ساختار فهرست زیر را در فهرست برنامه کاربردی سفارش ایجاد کنید:

```
locale/
  en/
  es/
```

با ایجاد یک فهرست محلی ، رشته های ترجمه این برنامه به جای فایل پیام اصلی ، در یک پرونده پیام تحت این فهرست ذخیره می شوند. به این روش ، می توانید برای هر برنامه ، فایل های ترجمه جدآگانه تولید کنید.

پوسته را از فهرست پروژه باز کنید و دستور زیر را اجرا کنید:

```
django-admin makemessages --all
```

باید خروجی زیر را مشاهده کنید:

```
processing locale es
processing locale en
```

پرونده محلی / es / LC_MESSAGES / django.po برنامه سفارش را با استفاده از ویرایشگر متن باز کنید. رشته های ترجمه را برای مدل Order مشاهده خواهید کرد. ترجمه های msgstr را برای رشته های داده شده در زیر پر کنید:

```
#: orders/models.py:10
msgid "first name"
```

```
msgstr "nombre"

#: orders/models.py:11
msgid "last name"
msgstr "apellidos"

#: orders/models.py:12
msgid "e-mail"
msgstr "e-mail"

#: orders/models.py:13
msgid "address"
msgstr "dirección"

#: orders/models.py:14
msgid "postal code"
msgstr "código postal"

#: orders/models.py:15
msgid "city"
msgstr "ciudad"
```

بعد از اتمام اضافه کردن ترجمه ها ، پرونده را ذخیره کنید.

علاوه بر ویرایشگر متن ، می توانید از Poedit برای ویرایش ترجمه ها نیز استفاده کنید. Poedit یک نرم افزار برای ویرایش ترجمه هاست و از gettext استفاده می کند. این نرم افزار برای لینوکس ، ویندوز و macOS در دسترس است. شما می توانید Poedit را از <https://poedit.net> بارگیری کنید. بیایید اشکال پروژه خود را نیز ترجمه کنیم. OrderCreateForm از درخواست سفارش لازم نیست ترجمه شود زیرا verbose_name ModelForm است و از ویژگی Order برای برچسب های زمینه فرم استفاده می کند. ما قصد داریم اشکال برنامه های سبد خرید و کوپن را از فیلدهای مدل Order ترجمه کنیم.

پرونده فرم.py را درون فهرست برنامه کاربردی سبد ویرایش کرده و یک ویژگی برچسب را به قسمت کمیت CartAddProductForm اضافه کنید ، و سپس این قسمت را برای ترجمه به شرح زیر علامت بزنید:

```
from django import forms
from django.utils.translation import gettext_lazy as _
```

```
PRODUCT_QUANTITY_CHOICES = [(i, str(i)) for i in range(1, 21)]

class CartAddProductForm(forms.Form):
    quantity = forms.TypedChoiceField(
        choices=PRODUCT_QUANTITY_CHOICES,
        coerce=int,
        label=_('Quantity'))
    update = forms.BooleanField(required=False,
        initial=False,
        widget=forms.HiddenInput)
```

فایل `form.py` کاربرد کوپن را ویرایش کنید و فرم `CouponApplyForm` را به شرح زیر ترجمه کنید:

```
from django import forms
from django.utils.translation import gettext_lazy as _

class CouponApplyForm(forms.Form):
    code = forms.CharField(label=_('Coupon'))
```

ما یک برچسب به قسمت کد اضافه کرده ایم و آن را برای ترجمه علامت گذاری کرده ایم.

ترجمه الگوهای

Django برای ترجمه رشته ها در قالب ها ، برچسب های قالب { % trans % } و { % blocktrans % } را ارائه می دهد. برای استفاده از برچسب های قالب ترجمه ، باید بارگیری کنید { % load i18n % } در بالای الگوی خود.

برچسب قالب { % trans % }: برچسب قالب { % trans % } به شما امکان می دهد یک رشته ، یک محتوای ثابت یا متغیر را برای ترجمه علامت گذاری کنید. در داخل ، جنگو متن متن داده شده را دریافت می کند. این

```
{% trans "Text to be translated" %}
```

می توانید از مطالب ترجمه شده در متغیری استفاده کنید که می توانید در تمام الگوی خود استفاده کنید. مثال زیر متن ترجمه شده را در متغیری به نام سلام (ذخیره) ذخیره می کند:

```
{% trans "Hello!" as greeting %}
<h1>{{ greeting }}</h1>
```

برچسب { % trans % } برای رشته های ترجمه ساده مفید است ، اما نمی تواند محتوا را برای ترجمه شامل متغیرها انجام دهد.

برچسب قالب { % blocktrans % }

برچسب قالب { % blocktrans % } به شما امکان می دهد محتوا را شامل کنید که شامل الفبای و محتوای متغیر با استفاده از متغیرهایی است. مثال زیر نحوه استفاده از برچسب { % blocktrans % } ، از جمله متغیر نام در محتوا را برای ترجمه نشان می دهد:

```
{% blocktrans %}Hello {{ name }}!{% endblocktrans %}
```

با استفاده از آنها می توانید عبارات الگویی مانند دسترسی به صفات شی یا استفاده از فیلترهای الگورا به متغیرها وارد کنید. شما همیشه باید از متغیرهایی برای این موارد استفاده کنید. شما نمی توانید به عبارات یا ویژگی های شی در بلوک blocktrans دسترسی پیدا کنید. مثال زیر نحوه استفاده از ویژگی های شی را که فیلتر capfirst در آن استفاده شده است نشان می دهد:

```
{% blocktrans with name=user.name|capfirst %}
    Hello {{ name }}!
{% endblocktrans %}
```

ترجمه قالب فروشگاه

قالب shop/base.html برنامه فروشگاه را ویرایش کنید. اطمینان حاصل کنید که برجسب 18n را در بالای الگوی بارگیری کرده و رشته ها را برای ترجمه به شرح زیر علامت گذاری می کنید:

```
{% load i18n %}  
{% load static %}  
<!DOCTYPE html>  
<html>  
<head>  
    <meta charset="utf-8" />  
    <title>  
        {% block title %}{% trans "My shop" %}{% endblock %}  
    </title>  
    <link href="{% static "css/base.css" %}" rel="stylesheet">  
</head>  
<body>  
    <div id="header">  
        <a href="/" class="logo">{% trans "My shop" %}</a>  
    </div>  
    <div id="subheader">  
        <div class="cart">  
            {% with total_items=cart|length %}  
            {% if cart|length > 0 %}  
                {% trans "Your cart" %}:  
                <a href="{% url "cart:cart_detail" %}">  
                    {% blocktrans with total_items_plural=total_items|pluralize  
                    total_price=cart.get_total_price %}  
                    {{ total_items }} item{{ total_items_plural }},  
                    ${{ total_price }}  
                {% endblocktrans %}  
                </a>  
            {% else %}  
                {% trans "Your cart is empty." %}  
            
```

```
        {% endif %}
        {% endwith %}
    </div>
</div>
<div id="content">
    {% block content %}
```

```
    {% endblock %}
</div>
</body>
</html>
```

برای نمایش خلاصه سبد ، به برجسب { % توجه کنید. خلاصه سبد خرید قبلاً به شرح زیر بود:

```
 {{ total_items }} item{{ total_items|pluralize }},
${{ cart.get_total_price }}
```

ما از { % برای تنظیم متغیرهایی برای ttotal_items|pluralize (برجسب الگوی مورد استفاده در اینجا) و سبد خرید. get_total_price (روش شیئی که در اینجا نامیده می شود) استفاده کردیم ، در نتیجه موارد زیر وجود دارد:

```
{% blocktrans with total_items_plural=total_items|pluralize
    total_price=cart.get_total_price %}
    {{ total_items }} item{{ total_items_plural }},
    ${{ total_price }}
{% endblocktrans %}
```

در مرحله بعد ، قالب shop/product/detail.html برنامه فروشگاه را ویرایش کنید و برچسب های i18n را در بالای آن بارگذاری کنید ، اما بعد از برچسب { % extends %} ، که همیشه باید اولین برچسب در این الگو باشد:

```
{% load i18n %}
```

سپس خط زیر را پیدا کنید:

```
<input type="submit" value="Add to cart">
```

آن را با موارد زیر جایگزین کنید:

```
<input type="submit" value="{% trans "Add to cart" %}">
```

اکنون ، الگوهای برنامه سفارشات را ترجمه کنید. قالب orders/order/create.html برنامه درخواست سفارش و متن علامت گذاری برای ترجمه را به شرح زیر ویرایش کنید:

```

{% extends "shop/base.html" %}
{% load i18n %}

{% block title %}
    {% trans "Checkout" %}
{% endblock %}

{% block content %}
<h1>{% trans "Checkout" %}</h1>

<div class="order-info">
    <h3>{% trans "Your order" %}</h3>
    <ul>
        {% for item in cart %}
            <li>
                {{ item.quantity }}x {{ item.product.name }}
                <span>${{ item.total_price }}</span>
            </li>
        {% endfor %}
        {% if cart.coupon %}
            <li>
                {% blocktrans with code=cart.coupon.code
                    discount=cart.coupon.discount %}
                    "{{ code }}" ({{ discount }}% off)
                {% endblocktrans %}
                <span>- ${{ cart.get_discount|floatformat:"2" }}</span>
            </li>
        {% endif %}
    </ul>
    <p>{% trans "Total" %}: ${{ cart.get_total_price_after_discount|floatformat:"2" }}</p>
</div>

<form action"." method="post" class="order-form">
    {{ form.as_p }}
    <p><input type="submit" value="{} trans "Place order" {}"></p>
    {% csrf_token %}
</form>
{% endblock %}

```

به کدهای همراه با این فصل نگاهی به پرونده های زیر بیندازید تا ببینید که چگونه رشته ها برای ترجمه مشخص شده اند:

The shop application: Template shop/product/list.html

The orders application: Template orders/order/created.html

The cart application: Template cart/detail.html

باید پرونده های پیام را به روز کنیم تا رشته های ترجمه جدید را در بر گیریم.

پوسته را باز کرده و دستور زیر را اجرا کنید:

```
django-admin makemessages --all
```

پرونده های po در داخل فهرست محلی پروژه Myshop قرار دارند و می بینید که برنامه سفارش اکنون شامل تمام رشته هایی است که ما برای ترجمه مشخص کرده ایم.

پرونده های ترجمه po پروژه و برنامه سفارش را ویرایش کنید و ترجمه های اسپانیایی را نیز در msgstr وارد کنید.

دستور زیر را برای کامپایل کردن فایل های ترجمه اجرا کنید:

```
django-admin compilemessages
```

خروجی زیر را مشاهده خواهید کرد:

```
processing file django.po in myshop/locale/en/LC_MESSAGES
processing file django.po in myshop/locale/es/LC_MESSAGES
processing file django.po in myshop/orders/locale/en/LC_MESSAGES
processing file django.po in myshop/orders/locale/es/LC_MESSAGES
```

یک پرونده .mo حاوی ترجمه های گردآوری شده برای هر پرونده ترجمه po تولید شده است.

با استفاده از رابط ترجمه Rosetta

یک برنامه شخص ثالث است که به شما امکان می دهد ترجمه ها را با استفاده از همان رابط کاربری سایت مدیریت جنگو ویرایش کنید. روزتا ویرایش فایل های po را آسان می کند و فایل های ترجمه شده کامپایل شده را به روز می کند. باید آن را به پروژه خود اضافه کنیم.

را از طریق پیپ با استفاده از این دستور نصب کنید:

```
pip install django-rosetta
```

سپس "پرونده" را در پرونده تنظیمات.py پروژه خود به تنظیمات INSTALLED_APPS اضافه کنید:

```
INSTALLED_APPS = [
    # ...
    'rosetta',
]
```

شما باید آدرس URL Rosetta را به پیکربندی اصلی URL خود اضافه کنید.

پرونده اصلی urls.py پروژه خود را ویرایش کنید و الگوی URL زیر را به آن اضافه کنید:

```
urlpatterns = [
    # ...
    path('rosetta/', include('rosetta.urls')),
    path('', include('shop.urls', namespace='shop')),
]
```

حتماً آن را قبل از الگوی shop.urls قرار دهید تا از الگوی ناخواسته تطابق نداشته باشد.

به <http://127.0.0.1:8000/admin> را باز کنید و با یک سرگرمی وارد شوید. سپس در مرورگر خود به <http://127.0.0.1:8000/rosetta> بروید. در فهرست فیلتر، روی قسمت سوم کلیک کنید تا تمام پرونده های پیام موجود از جمله پرونده هایی که به برنامه سفارش تعلق دارند، نمایش داده شود. شما باید لیستی از زبان های موجود را مشاهده کنید

به شرح زیر است:

Filter: PROJECT THIRD PARTY DjangO ALL

English

APPLICATION	PROGRESS	MESSAGES	TRANSLATED	FUZZY	OBsolete	FILE
Myshop	0%	12	0	0	0	/Users/zeno/dbe/myshop/locale/en/LC_MESSAGES/django.po
Orders	0%	13	0	0	0	/Users/zeno/dbe/myshop/orders/locale/en/LC_MESSAGES/django.po

Spanish

APPLICATION	PROGRESS	MESSAGES	TRANSLATED	FUZZY	OBsolete	FILE
Myshop	100%	12	12	0	0	/Users/zeno/dbe/myshop/locale/es/LC_MESSAGES/django.po
Orders	100%	13	13	0	0	/Users/zeno/dbe/myshop/orders/locale/es/LC_MESSAGES/django.po
Rosetta	73%	37	28	1	2	/Users/zeno/env/dbe3/lib/python3.6/site-

برای ویرایش ترجمه های اسپانیایی روی پیوند Myshop در زیر بخش اسپانیایی کلیک کنید. لیستی از رشته های ترجمه را باید به شرح زیر مشاهده کنید:

Translate into Spanish

Display: UNTRANSLATED ONLY TRANSLATED ONLY FUZZY ONLY ALL

ORIGINAL	SPANISH	FUZZY OCCURRENCE(S)
Quantity	Cantidad	<input type="checkbox"/> cart/forms.py:12
Coupon	Cupón	<input type="checkbox"/> coupons/forms.py:6
English	Ingles	<input type="checkbox"/> myshop/settings.py:117
Spanish	Español	<input type="checkbox"/> myshop/settings.py:118
My shop	Mi tienda	<input type="checkbox"/> shop/templates/shop/base.html:7 <input type="checkbox"/> shop/templates/shop/base.html:12
Your cart	Tu carro	<input type="checkbox"/> shop/templates/shop/base.html:18

می توانید ترجمه ها را در زیر ستون اسپانیایی وارد کنید. ستون (S) OCCURRENCES پرونده ها و خط کد را که در آن هر رشته ترجمه پیدا شده است نشان می دهد.

ترجمه هایی که شامل متغیرهای هستند به شرح زیر است:

%(total_items)s item%
(total_items_plural)s,
\$%(total_price)s

%(total_items)s producto%
(total_items_plural)s,
\$%(total_price)s

shop/templates/shop/base.html:20

روزتا از یک رنگ پس زمینه متفاوت برای نمایش متغیرهایی استفاده می کند.

هنگام ترجمه محتوا ، اطمینان حاصل کنید که متغیرهایی را ترجمه نکرده نگه دارید. به عنوان مثال رشته زیر را انتخاب کنید:

```
%(total_items)s item%(total_items_plural)s, $%(total_price)s
```

این ترجمه به زبان اسپانیایی به شرح زیر است:

```
%(total_items)s producto%(total_items_plural)s, $%(total_price)s
```

برای استفاده از همان ترجمه های اسپانیایی برای پروژه خود ، می توانید به کد منبع که همراه این فصل است نگاهی بیندازید.

پس از پایان ویرایش ترجمه ، روی دکمه ذخیره و ترجمه بلوک بعدی کلیک کنید تا ترجمه ها در پرونده po ذخیره شود. هنگام ذخیره ترجمه ، فایل پیغام را کامپایل می کند ، بنابراین نیازی به اجرای دستور compilemessages Rosetta شما نیست. با این حال ، Rosetta برای نوشتن پرونده های پیام نیاز به دسترسی به فهرستهای محلی دارد. مطمئن شوید که دایرکتوری ها دارای مجوزهای معتبر هستند.

اگر می خواهید سایر کاربران بتوانند ترجمه ها را ویرایش کنند ، http://127.0.0.1:8000/admin/auth/group/add در مرورگر خود باز کنید و گروه جدیدی به نام مترجمان ایجاد کنید. سپس به http://127.0.0.1:8000/admin/auth/user آورید ، ویرایش کنید تا آنها بتوانند ترجمه ها را ویرایش کنند. هنگام ویرایش یک کاربر ، در قسمت Permission ها ، گروه مترجمان را برای هر کاربر به گروه های انتخاب شده اضافه کنید. Rosetta فقط در اختیار superusers یا کاربرانی است که به گروه مترجمان تعلق دارند.

ترجمه های فازی

شاید متوجه شده باشید که یک ستون FUZZY در روزتا وجود دارد.

این یک ویژگی Rosetta نیست. این توسط `gettext` تهیه شده است. اگر پرچم فازی برای ترجمه فعال باشد ، در پروندهای پیام تلفیقی وارد نمی شود. این پرچم رشته های ترجمه را مشخص می کند که باید توسط یک مترجم بررسی شود. هنگامی که پروندهای `po` با رشته های ترجمه جدید به روز می شوند ، ممکن است که برخی از رشته های ترجمه به صورت خودکار فازی پرچم گذاری شوند. این اتفاق می افتد وقتی `gettext` بعضی از `msgid` را پیدا کند که کمی تغییر یافته باشد. `gettext` آن را با آنچه که ترجمه قدیمی ترجمه می کند جفت می کند و آن را به عنوان فازی برای بررسی پرچم می کند. سپس مترجم باید ترجمه های فازی را بررسی کند ، پرچم فازی را برداشته و فایل ترجمه را مجدداً گردآوری کند.

الگوهای URL برای بین المللی

قابلیت بین المللی سازی را برای URL ها ارائه می دهد. این شامل دو ویژگی اصلی برای URL های بین المللی است:

پیشوند زیان در الگوهای URL: اضافه کردن یک پیشوند زیان به URL ها برای ارائه هر نسخه زیان تحت یک URL پایه دیگر: الگوهای URL ترجمه شده: ترجمه الگوهای URL به گونه ای که هر URL برای هر زبان متفاوت است ، یک دلیل برای ترجمه URL ها بهینه سازی سایت شما برای جستجو است. موتورها با افزودن یک پیشوند زیان به الگوهای خود ، می توانند به جای یک URL واحد برای همه آنها ، URL را برای هر زبان فهرست کنند. علاوه بر این ، با ترجمه URL به هر زبان ، آدرسهای URL را به موتورهای جستجو ارائه می دهید که برای هر زیان رتبه بهتری دارند.

افزودن یک پیشوند زیان به الگوهای URL

به شما امکان می دهد یک پیشوند زیان را به الگوهای URL خود اضافه کنید.

به عنوان مثال ، نسخه انگلیسی سایت شما می تواند تحت مسیری از `/en/` ، و نسخه اسپانیایی `/es/` ارائه شود.

برای استفاده از زیانها در الگوهای URL ، باید از `LocaleMiddleware` استفاده شده توسط Django استفاده کنید. این چارچوب از آن برای شناسایی زبان فعلی از URL درخواستی استفاده می کند. شما قبل از آن را به تنظیمات `MiddleWARE` پردازش خود اضافه کرده اید ، بنابراین دیگر نیازی به انجام آن ندارید.

باید یک پیشوند زبان به الگوهای URL ما اضافه کنیم. پرونده urls.py اصلی پروژه myshop را ویرایش کنید و `i18n_patterns()` را به شرح زیر اضافه کنید:

```
from django.conf.urls.i18n import i18n_patterns

urlpatterns = i18n_patterns(
    path('admin/', admin.site.urls),
    path('cart/', include('cart.urls', namespace='cart')),
    path('orders/', include('orders.urls', namespace='orders')),
    path('payment/', include('payment.urls', namespace='payment')),
    path('coupons/', include('coupons.urls', namespace='coupons')),
    path('rosetta/', include('rosetta.urls')),
    path('', include('shop.urls', namespace='shop')),
)
```

می توانید الگوهای و الگوهای استاندارد غیر قابل ترجمه را در زیر `i18n_patterns()` ترکیب کنید تا برخی از الگوهای شامل پیشوند زبان باشند و برخی دیگر اینگونه نیستند. با این حال ، بهتر است از URL های ترجمه شده فقط استفاده کنید تا از احتمال عدم دسترسی به آدرس اینترنتی ترجمه شده با الگوی URL ترجمه نشده ، استفاده نشود.

سرور توسعه را اجرا کنید و `http://127.0.0.1:8000/` را در مرورگر خود باز کنید. مراحلی را که قبلًا در چگونه Django برای تعیین زبان فعلی تعیین می کند ، بخش زبان فعلی را تعیین می کند ، و شما را به URL درخواست شده از جمله پیشوند زیان هدایت می کند. به URL در مرورگر خود نگاهی بیندازید. اکنون باید شبیه `http://127.0.0.1:8000/fa/` باشد.

اگر زبان اسپانیایی یا انگلیسی باشد ، زبان فعلی یک زبان تنظیم شده توسط عنوان `Accept-Language` در مرورگر شما است ، در غیر این صورت `LANGUAGE_CODE` (انگلیسی) پیش فرض در تنظیمات شما تنظیم شده است.

ترجمه الگوهای URL

جنگو از رشته های ترجمه شده در الگوهای URL پشتیبانی می کند. برای هر الگوی URL می توانید از ترجمه دیگری برای هر زیان استفاده کنید.

با استفاده از عملکرد `gettext_lazy()` می توانید الگوهای URL را برای ترجمه به همان روشی که با ادبیات انجام می دهد علامت گذاری کنید.

پرونده urls.py اصلی پروژه myshop را ویرایش کنید و رشته های ترجمه را به عبارات منظم الگوهای URL مربوط به برنامه های سبد خرید ، سفارشات ، پرداخت و کوین ها به شرح زیر اضافه کنید:

```
from django.utils.translation import gettext_lazy as _

urlpatterns = i18n_patterns(
    path(_('admin/'), admin.site.urls),
    path(_('cart/'), include('cart.urls', namespace='cart')),
    path(_('orders/'), include('orders.urls', namespace='orders')),
    path(_('payment/'), include('payment.urls', namespace='payment')),
    path(_('coupons/'), include('coupons.urls', namespace='coupons')),
    path('rosetta/', include('rosetta.urls')),
    path('', include('shop.urls', namespace='shop')),
)
```

فایل urls.py از برنامه سفارشات را ویرایش کنید و الگوهای URL را برای ترجمه به شرح زیر علامت گذاری کنید:

```
from django.utils.translation import gettext_lazy as _

urlpatterns = [
    path(_('create/'), views.order_create, name='order_create'),
    # ...
]
```

پرونده urls.py برنامه پرداخت را ویرایش کنید و کد را به شرح زیر تغییر دهید:

```
from django.utils.translation import gettext_lazy as _

urlpatterns = [
    path(_('process/'), views.payment_process, name='process'),
    path(_('done/'), views.payment_done, name='done'),
    path(_('canceled/'), views.payment_canceled, name='canceled'),
]
```

ما نیازی به ترجمه الگوهای URL برنامه فروشگاه نداریم ، زیرا آنها با متغیرهای ساخته شده اند و شامل هیچگونه ادبی دیگری نیستند.

پوسته را باز کنید و دستور بعدی را برای به روزرسانی پرونده های پیام با ترجمه های جدید اجرا کنید:

```
django-admin makemessages --all
```

اطمینان حاصل کنید که سرور توسعه در حال اجرا است. <http://127.0.0.1:8000/en/rosetta/> را در مرورگر خود باز کنید و روی لینک Myshop در قسمت اسپانیایی کلیک کنید. اکنون الگوهای URL برای ترجمه را مشاهده خواهید کرد. فقط می توانید روی Untranslated کلیک کنید تا فقط رشته هایی را ترجمه کنید که هنوز ترجمه نشده اند. اکنون می توانید آدرس های اینترنتی را ترجمه کنید.

اجازه دادن به کاربران برای تغییر زبان

از آنجاکه ما در حال ارائه مطالی هستیم که به چندین زبان موجود است ، باید به کاربران خود اجازه دهیم که زبان سایت را تغییر دهند. ما قصد داریم یک انتخاب زبان را به سایت خود اضافه کنیم. انتخاب زبان از لیستی از زبان های موجود تشکیل می شود ، که با استفاده از لینک ها نمایش داده می شوند.

قالب shop/base.html برنامه فروشگاه را ویرایش کنید و خطوط زیر را پیدا کنید:

```
<div id="header">
  <a href="/" class="logo">[% trans "My shop" %]</a>
</div>
```

آنها را با کد زیر جایگزین کنید:

```

<div id="header">
  <a href="/" class="logo">{% trans "My shop" %}</a>

  {% get_current_language as LANGUAGE_CODE %}
  {% get_available_languages as LANGUAGES %}
  {% get_language_info_list for LANGUAGES as languages %}
  <div class="languages">
    <p>{% trans "Language" %}:</p>
    <ul class="languages">
      {% for language in languages %}
        <li>
          <a href="/{{ language.code }}/" 
            {% if language.code == LANGUAGE_CODE %} class="selected"{% endif %}
          {{ language.name_local }}
          </a>
        </li>
      {% endfor %}
    
```

```

      </ul>
    </div>
  </div>

```

نحوه انتخاب زبان ما این است:

1. ابتدا برچسب های بین المللی سازی را با استفاده از load%load i18n load%load بارگذاری می کنیم.
2. ما برای بازیابی زیان فعلی از برچسب { %get_current_language %} استفاده می کنیم
3. ما با استفاده از برچسب قالب { %get_available_languages } the زبانهای را که در تنظیمات LANGUAGES تعريف شده اند ، دریافت می کنیم.
4. ما از تگ { %get_language_info_list %} برای دسترسی آسان به ویژگی های زبان استفاده می کنیم
- 5- ما یک لیست HTML ایجاد می کنیم تا کلیه زیان های موجود را نمایش دهیم و یک ویژگی کلاس انتخاب شده را به زیان فعال فعلی اضافه می کنیم از برچسب های الگوی ارائه شده توسط i18n ، بر اساس زیان های موجود در تنظیمات پروژه شما استفاده می کنیم.

اکنون در مرورگر خود <http://127.0.0.1:8000/> را باز کنید و نگاهی بیندازید. شما باید انتخاب کننده زبان را در گوشه سمت راست بالای سایت به شرح زیر مشاهده کنید:

Mitienda

Language: English Español

Tu carro está vacío.

Productos

Categorías

Todos

Tea

NO IMAGE AVAILABLE



Green tea \$30

Red tea \$45,5

Tea powder \$21,2

کاربران اکنون می توانند به راحتی به زبان دلخواه خود تغییر دهند.

ترجمه مدل‌ها با django-parler

جنگو راه حلی برای ترجمه مدل‌های خارج از جعبه ارائه نمی‌دهد. شما باید راه حل خود را برای مدیریت محتوای ذخیره شده در زبانهای مختلف پیاده کنید یا از یک ماژول شخص ثالث برای ترجمه مدل استفاده کنید. چندین برنامه شخص ثالث وجود دارد که به شما امکان می‌دهد زمینه‌های مدل را ترجمه کنید. هرکدام از آنها برای ذخیره و دسترسی به ترجمه‌ها رویکرد متفاوتی را اتخاذ می‌کنند. یکی از این برنامه‌ها django-parler است. این ماژول روشی بسیار مؤثر برای ترجمه مدل‌ها ارائه می‌دهد و هموار با سایت مدیریت Django یکپارچه می‌شود.

django-parler برای هر مدل جدول پایگاه داده جداگانه‌ای تولید می‌کند که شامل ترجمه‌هاست. این جدول شامل تمام قسمت‌های ترجمه شده و یک کلید خارجی برای شی اصلی است که ترجمه به آن تعلق دارد. همچنین شامل یک قسمت زیان است، زیرا هر سطر محتوا را برای یک زبان واحد ذخیره می‌کند.

نصب django-parler

django-parler را از طریق پیپ با استفاده از دستور زیر نصب کنید:

```
pip install django-parler
```

پرونده settings.py پروژه خود را ویرایش کنید و 'تنظیم' را به تنظیمات INSTALLED_APPS به شرح زیر اضافه کنید:

```
INSTALLED_APPS = [
    # ...
    'parler',
]
```

همچنین کد زیر را به تنظیمات خود اضافه کنید:

```
PARLER_LANGUAGES = {
    None: (
        {'code': 'en'},
        {'code': 'es'},
    ),
    'default': {
        'fallback': 'en',
        'hide_untranslated': False,
    }
}
```

این تنظیم زیان های موجود و اصلی را برای django-parler تعریف می کند.

ما زیان پیش فرض را مشخص می کنیم و django-parler را نشان می دهیم، باید محتوای ترجمه نشده را پنهان کند.

زمینه های مدل ترجمه

باید ترجمه هایی را برای کاتالوگ محصولات خود اضافه کنیم. django-parler یک کلاس مدل TranslatedModel و TranslatedFields برای ترجمه زمینه های مدل فراهم می کند. پرونده model.py را درون دایرکتوری برنامه فروشگاه ویرایش کرده و واردات زیر را اضافه کنید:

```
from parler.models import TranslatableModel, TranslatedFields
```

سپس مدل Category را اصلاح کنید تا نام و فیلدهای Slug به صورت زیر قابل ترجمه باشد:

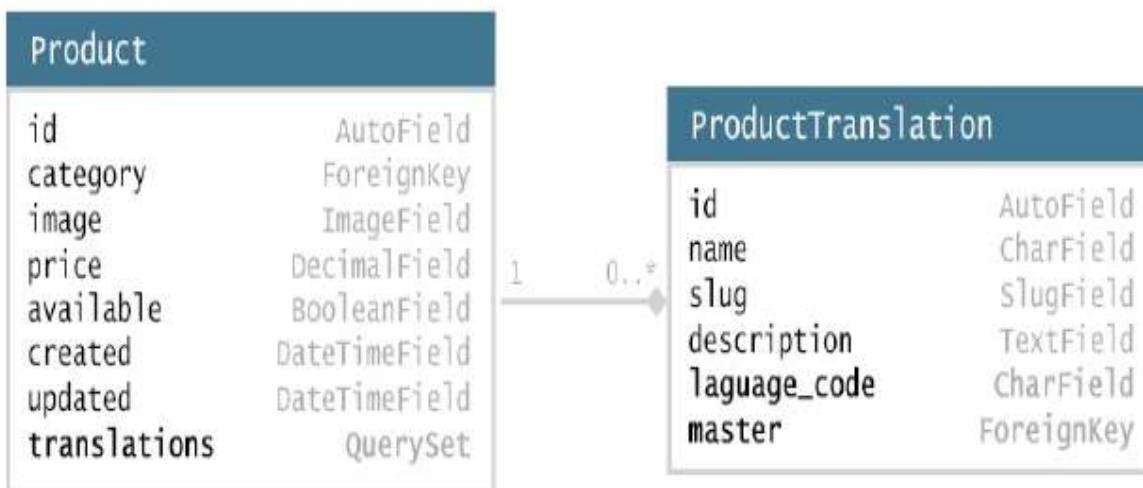
```
class Category(TranslatableModel):
    translations = TranslatedFields(
        name = models.CharField(max_length=200,
                               db_index=True),
        slug = models.SlugField(max_length=200,
                               db_index=True,
                               unique=True)
    )
```

مدل طبقه اکنون به جای مدلها از TranslatedModel به ارث می برد. زمینه و مدل و هر دو نام slug در بسته بندی TranslatedFields گنجانده شده است.

مدل محصول را ویرایش کنید تا ترجمه هایی را برای قسمت های نام ، slug و توضیحات به شرح زیر اضافه کنید:

```
class Product(TranslatableModel):
    translations = TranslatedFields(
        name = models.CharField(max_length=200, db_index=True),
        slug = models.SlugField(max_length=200, db_index=True),
        description = models.TextField(blank=True)
    )
    category = models.ForeignKey(Category,
                                 related_name='products')
    image = models.ImageField(upload_to='products/%Y/%m/%d',
                             blank=True)
    price = models.DecimalField(max_digits=10, decimal_places=2)
    available = models.BooleanField(default=True)
    created = models.DateTimeField(auto_now_add=True)
    updated = models.DateTimeField(auto_now=True)
```

ترجمه ها را با تولید مدل دیگری برای هر مدل قابل ترجمه مدیریت می کند. در ادامه، شما می توانید زمینه های مدل محصول و چه مدل ProductTranslation تولید خواهد شد مثال را ببینید:



مدل ProductTranslation تولید شده توسط django-parler شامل `name` ، `slug` ، `description` و `language_code` است. یک رابطه یک به یک از محصول تا ProductTranslation وجود دارد.

یک شیء ProductTranslation برای هر زبان موجود برای هر شیء محصول وجود خواهد داشت. از آنجاکه جنگو از یک جدول جداگانه برای ترجمه استفاده می کند، برخی از ویژگی های جنگو وجود دارد که نمی توانیم از آنها استفاده کنیم. استفاده از یک سفارش پیش فرض توسط یک قسمت ترجمه شده امکان پذیر نیست. می توانید با استفاده از فیلدهای ترجمه شده در نمایش داده ها فیلتر کنید، اما نمی توانید یک قسمت قابل ترجمه را در گزینه های متا سفارش دهید.

پرونده `model.py` برنامه فروشگاه را ویرایش کنید و درباره ویژگی سفارش کلاس متا طبقه بندی اظهار نظر کنید:

```
class Category(TranslatableModel):
    # ...
    class Meta:
        # ordering = ('name',)
        verbose_name = 'category'
        verbose_name_plural = 'categories'
```

ما همچنین باید در مورد سفارش و ویژگی های Product Meta کلاس index_together اظهار نظر کنیم. نسخه فعلی پشتیبانی برای تأیید اعتبار index_together ارائه نمی دهد. کلاس متأ مخصوص را به شرح زیر اعلام کنید:

```
class Product(TranslatableModel):
    # ...

    # class Meta:
    #     ordering = ('-name',)
    #     index_together = (('id', 'slug'),)
```

می توانید اطلاعات بیشتری در مورد سازگاری ماژول django-parler با Django در اینجا بخوانید

<https://django-parler.readthedocs.io/fa/latest/compatibility.html>

ادغام ترجمه ها در سایت مدیریت

ترجمه های ModelAdmin با سایت مدیریت جنگو هموار است.

این شامل یک کلاس TranslatableAdmin است که کلاس ModelAdmin ارائه شده توسط Django برای مدیریت ترجمه های مدل را نادیده می گیرد.

فایل admin.py برنامه فروشگاه را ویرایش کنید و واردات زیر را به آن اضافه کنید:

```
from parler.admin import TranslatableAdmin
```

کلاس‌های ProductAdmin و ClassAdmin را اصلاح کنید تا به جای ModelAdmin از وراثت TranslatableAdmin به ارت برید. django-parler پشتیبانی نمی‌کند، اما از این پشتیبانی می‌کند () متد که عملکرد مشابهی را ارائه می‌دهد.

باید این را بر این اساس تغییر دهیم. پرونده admin.py را ویرایش کنید تا به صورت زیر ظاهر شود:

```
from django.contrib import admin
from .models import Category, Product
from parler.admin import TranslatableAdmin

@admin.register(Category)
class CategoryAdmin(TranslatableAdmin):
    list_display = ['name', 'slug']

    def get_prepopulated_fields(self, request, obj=None):
        return {'slug': ('name',)}


@admin.register(Product)
class ProductAdmin(TranslatableAdmin):
    list_display = ['name', 'slug', 'price',
                    'available', 'created', 'updated']
    list_filter = ['available', 'created', 'updated']

    list_editable = ['price', 'available']

    def get_prepopulated_fields(self, request, obj=None):
        return {'slug': ('name',)}
```

ما سایت مدیریت را برای کار با مدل‌های جدید ترجمه شده سازگار کرده‌ایم. اکنون می‌توانیم دیتابیس را با مدل تغییراتی که ساخته ایم همگام سازی کنیم.

ایجاد مهاجرت برای ترجمه مدل

پوسته را باز کرده و دستور زیر را اجرا کنید تا یک انتقال جدید برای ترجمه مدل ایجاد شود:

```
python manage.py makemigrations shop --name "translations"
```

خروجی زیر را مشاهده خواهید کرد:

```
Migrations for 'shop':
  shop/migrations/0002_translations.py
    - Create model CategoryTranslation
    - Create model ProductTranslation
    - Change Meta options on category
    - Change Meta options on product
    - Remove field name from category
    - Remove field slug from category
    - Alter index_together for product (0 constraint(s))
    - Add field master to producttranslation
    - Add field master to categorytranslation
    - Remove field description from product
    - Remove field name from product
    - Remove field slug from product
    - Alter unique_together for producttranslation (1 constraint(s))
    - Alter unique_together for categorytranslation (1 constraint(s))
```

این مهاجرت بطور خودکار شامل مدل های طبقه بندی و انتقال محصول و ترانزیشن است که به صورت پویا توسط django-parler ایجاد شده است. توجه به این نکته ضروری است که این مهاجرت قسمتهای قبلی موجود را از مدل های ما حذف می کند. این بدان معنی است که ما آن داده ها را از دست خواهیم داد و لازم است گروه ها و محصولات خود را بعد از اجرای مجددا در سایت مدیر قرار دهیم.

دستور زیر را برای اعمال مهاجرت اجرا کنید:

```
python manage.py migrate shop
```

خروجی را خواهید دید که با خط زیر پایان می یابد:

```
Applying shop.0002_translations... OK
```

مدلهای ما اکنون با بانک اطلاعات همگام شده اند.

سرور توسعه را با استفاده از serverth python management.py اجرا کنید و <http://127.0.0.1:8000/en/admin/shop/categ> را در مرورگر خود باز کنید. خواهید دید که گروههای موجود به دلیل حذف آن قسمتها و استفاده از مدلها قابل ترجمه تولید شده توسط djangoparler ، نام و اسلوب خود را از دست داده اند. برای ویرایش آن بر روی یک دسته کلیک کنید. خواهید دید که

صفحه تغییر دسته شامل دو زبانه متفاوت ، یک برای انگلیسی و دیگری برای ترجمه های اسپانیایی:

The screenshot shows the Django admin interface for a 'Category' model. The top navigation bar displays 'Django administration' and 'WELCOME, ADMIN. VIEW SITE / CHANGE PASSWORD / LOG OUT'. Below the navigation, a breadcrumb trail shows 'Home > Shop > Categories > Tea'. The main title is 'Change category (English)'. There are two tabs: 'English' (selected) and 'Spanish'. The form contains two fields: 'Name' with the value 'Tea' and 'Slug' with the value 'tea'. At the bottom are four buttons: 'Delete' (red), 'Save and add another' (light blue), 'Save and continue editing' (light blue), and 'SAVE' (dark blue).

اطمینان حاصل کنید که برای همه دسته های موجود یک نام و اسلحه بزنید.

همچنین برای هر یک از آنها ترجمه اسپانیایی اضافه کنید و بر روی دکمه SAVE کلیک کنید. مطمئن شوید که قبل از تغییر برگه ها ، تغییرات را ذخیره می کنید یا آنها را گم می کنید.

پس از تکمیل داده ها برای دسته های موجود ، <http://127.0.0.1:8000/en/admin/shop/product/> را باز کنید و هر کدام از موارد را ویرایش کنید

محصولات ارائه شده به اسم انگلیسی ، اسپانیایی ، اسلحه و توضیحات.

تطبیق نمایش برای ترجمه

برای استفاده از QuerySets ترجمه باید مجبور شویم نمای فروشگاه خود را تطبیق دهیم. دستور زیر را برای باز کردن پوسته پایتون اجرا کنید:

```
python manage.py shell
```

باید نگاهی بیندازیم که چگونه می توانید زمینه های ترجمه را بازیابی و پرس و جو کنید. برای بدست آوردن شی با فیلدهای قابل ترجمه به یک زبان خاص ، می توانید از عملکرد `Django active()` به شرح زیر استفاده کنید:

```
>>> from shop.models import Product
>>> from django.utils.translation import activate
>>> activate('es')
>>> product=Product.objects.first()
>>> product.name
'Té verde'
```

راه دیگر برای انجام این کار با استفاده از مدیر زبان ارائه شده توسط `django-parler` به شرح زیر است:

```
>>> product=Product.objects.language('en').first()
>>> product.name
'Green tea'
```

وقتی به قسمتهای ترجمه شده دسترسی پیدا می کنید ، آنها با استفاده از زیان فعلی برطرف می شوند. می توانید یک زبان فعلی متفاوت را برای یک شی تنظیم کنید تا به آن ترجمه خاص به شرح زیر دسترسی پیدا کند:

```
>>> product.set_current_language('es')
>>> product.name
'Té verde'
>>> product.get_current_language()
'es'
```

هنگام انجام `QuerySet` با استفاده از `filter()` ، می توانید با استفاده از اشیاء ترجمه مرتبط با `translations__ syntax` زیر فیلتر کنید:

```
>>> Product.objects.filter(translations__name='Green tea')
<TranslatableQuerySet [<Product: Té verde>]>
```

باید نمایش کاتالوگ محصول را تطبیق دهیم. پرونده views.py برنامه فروشگاه را ویرایش کنید و در نمای product_list خط زیر را پیدا کنید:

```
category = get_object_or_404(Category, slug=category_slug)
```

آن را با موارد زیر جایگزین کنید:

```
language = request.LANGUAGE_CODE
category = get_object_or_404(Category,
                             translations__language_code=language,
                             translations__slug=category_slug)
```

سپس نمای Product_detail را ویرایش کرده و خطوط زیر را پیدا کنید:

```
product = get_object_or_404(Product,
                            id=id,
                            slug=slug,
                            available=True)
```

نمایش Product_list و Product_detail اکنون برای بازیابی اشیاء با استفاده از فیلدهای ترجمه شده سازگار شده است. سرور توسعه را اجرا کنید و <http://127.0.0.1:8000/es/> را در مرورگر خود باز کنید. شما باید صفحه لیست محصولات، از جمله تمام محصولات ترجمه شده به اسپانیایی را مشاهده کنید:

Tu carro está vacío.

Productos

Categorías

Todos

Té

NO IMAGE
AVAILABLE

Té verde

\$30

Té rojo

\$45,5

Té en polvo

\$21,2

اکنون URL هر محصول با استفاده از قسمت Slug درست شده به زبان فعلی ساخته شده است. به عنوان مثال ، URL برای یک محصول به زبان اسپانیایی /http://127.0.0.1:8000/es/2/te-rojo/ است ، در حالی که به زبان انگلیسی URL /http://127.0.0.1:8000/en/2/red-tea/ به صفحه جزئیات محصول بروید ، همانطور که در مثال زیر نشان داده شده است ، URL ترجمه شده و محتوای زبان انتخاب شده را مشاهده می کنید:

Tu carro está vacío.



Té rojo

Té

\$45,5

Cantidad: 1

Añadir al carro

El té pu-erh es conocido en Occidente también como té rojo (en chino: 普洱茶, pinyin: pǔ'érchá) y su nombre proviene de la región de Pu'er de Yunnan China, de donde procede. Se trata de un té inusual en China, siendo este el mayor productor del té rojo o pu-erh del mundo.

اگر می خواهید درباره django-parler اطلاعات بیشتری کسب کنید ، می توانید مستندات کامل را در <https://django-parler.readthedocs.io/fa/latest> پیدا کنید.

شما یاد گرفته اید که چگونه کد پایتون ، الگوهای URL و زمینه های مدل را ترجمه کنید. برای تکمیل فرایند بین المللی و بومی سازی ، باید از قالب بندی محلی برای تاریخ ها ، زمان ها و اعداد نیز استفاده کنیم.

بومی سازی قالب

بسته به محل استفاده کاربر ، ممکن است بخواهید تاریخ ها ، زمان ها و اعداد را در قالب های مختلف نمایش دهید. قالب بندی محلی شده با تغییر تنظیمات USE_L10N در پرونده در تنظیمات.py پروژه شما می تواند فعال شود.

هنگامی که USE_L10N فعال باشد ، Django سعی خواهد کرد هر زمان که یک مقدار را در یک الگوی تولید کند ، از یک قالب محلی استفاده کند. می توانید ببینید که اعداد اعشاری در نسخه انگلیسی سایت شما با یک نقطه جداگانه برای مکان های اعشاری نمایش داده می شوند ، در حالی که در نسخه اسپانیایی با استفاده از کاما نمایش داده می شوند. این امر به دلیل

قالب‌های محلی است که توسط Django برای es locale مشخص شده است. می‌توانید به پیکربندی قالب بندی اسپانیایی در نگاهی <https://github.com/django/django/blob/stable/2.0.x/django/conf/locale/es/formats.py> بیندازید.

به طور معمول ، تنظیمات USE_L10N را روی True قرار می‌دهید و به Django اجازه می‌دهید محلی سازی فرمت را برای هر محلی ایجاد کند. با این وجود ، ممکن است موقعیت‌هایی باشد که شما نمی‌خواهید از مقادیر بومی شده استفاده کنید. این امر به ویژه هنگام خروج JSON یا JavaScript را می‌دهد ، مرتبط است.

یک localize %} قالب ارائه می‌دهد که به شما امکان می‌دهد محلی سازی قطعات قالب را روشن یا خاموش کنید. این به شما امکان می‌دهد تا بر فرمت بندی محلی شده کنترل کنید. برای استفاده از این برچسب قالب باید برچسب های l10n را بارگیری کنید. در زیر نمونه‌ای از چگونگی روشن و غیرفعال کردن بومی سازی در یک الگوی:

```
{% load l10n %}

{% localize on %}
  {{ value }}
{% endlocalize %}
```

```
{% localize off %}
  {{ value }}
{% endlocalize %}
```

Django همچنین فیلترهای قالب را بومی سازی و غیرمجاز کردن برای مجبور یا جلوگیری از محلی سازی یک مقدار ارائه می‌دهد. این فیلترها به شرح زیر قابل استفاده هستند:

```
 {{ value|localize }}  
 {{ value|unlocalize }}
```

همچنین می‌توانید برای مشخص کردن قالب بندی محلی، فایلهای با فرمت دلخواه ایجاد کنید. می‌توانید اطلاعات بیشتر در مورد محلی سازی قالب را در اینجا بباید.

<https://docs.djangoproject.com/fa/2.0/topics/i18n/formatting>

استفاده از django-localflavor برای تأیید صحیت زمینه‌های فرم django-localflavor یک ماژول شخص ثالث است که شامل مجموعه‌ای از ابزارهای خاص مانند فیلدهای فرم یا فیلدهای مدل است که مخصوص هر کشور است. اعتبار بخشیدن به مناطق محلی، شماره تلفن‌های محلی، شماره شناسنامه، شماره‌های تأمین اجتماعی و غیره بسیار مفید است. این بسته در یک سری ماژول‌ها با نام کد کشور ISO 3166 قرار گرفته است. django-localflavor را با استفاده از دستور زیر نصب کنید:

```
pip install django-localflavor==2.0
```

پرونده تنظیمات.py پروژه خود را ویرایش کرده و فلوئور محلی را به تنظیمات INSTALLED_APPS اضافه کنید:

```
INSTALLED_APPS = [  
    # ...  
    'localflavor',  
]
```

ما می‌خواهیم قسمت کد پستی ایالات متحده را اضافه کنیم تا یک کد پستی معتبر در ایالات متحده برای ایجاد سفارش جدید لازم باشد.

پرونده فرم.py برنامه سفارش را ویرایش کنید و به صورت زیر ظاهر کنید:

```
from django import forms
from .models import Order
from localflavor.us.forms import USZipCodeField
```

```
class OrderCreateForm(forms.ModelForm):
```

```
    postal_code = USZipCodeField()
    class Meta:
        model = Order
        fields = ['first_name', 'last_name', 'email', 'address',
                  'postal_code', 'city']
```

ما قسمت USZipCodeField را از بسته usl محلی ما وارد می کنیم و از آن برای قسمت postal_code از فرم OrderCreateForm استفاده می کنیم.

سرور توسعه را اجرا کنید و <http://127.0.0.1:8000/en/order/create> را در مرورگر خود باز کنید. همه قسمت ها را پر کنید، کد پسی سه حرف را وارد کنید و سپس فرم را ارسال کنید. خطای اعتبار سنجی زیر را که توسط USZipCodeField ایجاد شده است دریافت خواهید کرد:

```
Enter a zip code in the format XXXXX or XXXXX-XXXX.
```

این فقط یک نمونه کوتاه از نحوه استفاده از یک زمینه سفارشی از localflavor در پروژه خود برای اهداف اعتبار سنجی است. اجزای محلی ارائه شده توسط localflavor برای تطبیق برنامه شما با کشورهای خاص بسیار مفید است. می توانید django-localflavor را بخوانید

مستندات را مشاهده کرده و کلیه مؤلفه های محلی موجود برای هر کشور را در <https://django-localeflavor.readthedocs.io/fa/latest> مشاهده کنید.

در مرحله بعد ، ما قصد داریم موتور پیشنهادی را وارد مغازه خود کنیم.

ساختن موتور توصیه ای

موتور توصیه ای سیستمی است که ترجیح یا رتبه ای را که کاربر به یک مورد می دهد پیش بینی می کند. این سیستم بر اساس رفتار و دانش آنها در مورد آنها مواردی را برای کاربران انتخاب می کند. امروزه در بسیاری از خدمات آنلاین از سیستمهای توصیه استفاده می شود. آنها با انتخاب مطالب مورد علاقه خود از میان تعداد زیادی از داده های موجود که برای آنها بی ربط است ، به کاربران کمک می کنند. ارائه توصیه های خوب باعث افزایش تعامل کاربر می شود. سایت های تجارت الکترونیک همچنین با افزایش متوسط فروش خود ، از ارائه توصیه های مربوط به محصول سود می بزند.

ما قصد داریم یک موتور توصیه ای ساده اما در عین حال قدرتمند ایجاد کنیم که محصولاتی را پیشنهاد می کند که معمولاً با هم خریداری می شوند. ما محصولاتی را بر اساس فروش تاریخی پیشنهاد خواهیم کرد ، بنابراین کالاهایی را که معمولاً با هم خریداری می شوند ، شناسایی خواهیم کرد. ما قصد داریم محصولات مکمل را در دو سناریوی مختلف پیشنهاد کنیم:

صفحه جزئیات محصول: ما لیستی از محصولاتی را که معمولاً با محصول معین خریداری می شود ، نمایش خواهیم داد. به این ترتیب نمایش داده می شود: کاربرانی که این مورد را خریداری کرده اند ، X ، Y ، Z را نیز خریداری کرده اند. ما به ساختار داده ای نیاز داریم که به ما امکان می دهد تعداد دفعات خریداری شده توسط هر محصول را با کالای نمایش داده شده ، ذخیره کنیم.

صفحه جزئیات سبد خرید: بر اساس افزودن کاربران به سبد خرید ، قصد داریم کالاهایی را که معمولاً همراه این محصولات خریداری می شوند ، پیشنهاد کنیم. در این حالت ، امتیازی که برای به دست آوردن محصولات مرتبط محاسبه می کنیم باید جمع شود. ما قصد داریم از Redis برای ذخیره کالاهایی که در کنار هم خریداری می شوند استفاده کنیم. به یاد داشته باشید که قبل از Redis در فصل 6 ، پیگیری عملکردهای کاربر استفاده کرده اید. اگر هنوز Redis را نصب نکرده اید ، می توانید دستورالعمل های نصب را در آن فصل پیدا کنید.

توصیه محصولات بر اساس خریدهای قبلی

حال ما محصولات خود را بر اساس آنچه که به سبد خرید اضافه کرده اند ، به کاربران توصیه خواهیم کرد. ما قصد داریم برای هر محصول خریداری شده در سایت ما ، یک کلید در Redis ذخیره کنیم. کلید محصول شامل مجموعه ای مرتب شده با نمرات خواهد بود. ما هر بار خرید جدید با هم امتیاز را با 1 نمره افزایش می دهیم.

هنگامی که یک سفارش با موفقیت پرداخت می شود ، ما برای هر کالای خریداری شده یک کلید ، از جمله مجموعه ای از محصولات که به همان سفارش تعلق دارند ، ذخیره می کنیم. مجموعه مرتب شده به ما امکان می دهد برای محصولاتی که با هم خریداری می شوند امتیاز دهیم.

به یاد داشته باشید که با استفاده از دستور زیر ، redis-py را در محیط خود نصب کنید:

```
pip install redis
```

پرونده settings.py پروژه خود را ویرایش کنید و تنظیمات زیر را به آن اضافه کنید:

```
REDIS_HOST = 'localhost'  
REDIS_PORT = 6379  
REDIS_DB = 1
```

این تنظیمات مورد نیاز برای ایجاد ارتباط با سرور Redis است. یک پرونده جدید را درون فهرست برنامه کاربردی فروشگاه ایجاد کرده و آن را rekomander.py بنامید. کد زیر را به آن اضافه کنید:

```
import redis  
from django.conf import settings
```

```

from .models import Product

# connect to redis
r = redis.StrictRedis(host=settings.REDIS_HOST,
                      port=settings.REDIS_PORT,
                      db=settings.REDIS_DB)

class Recommender(object):

    def get_product_key(self, id):
        return 'product:{}:purchased_with'.format(id)

    def products_bought(self, products):
        product_ids = [p.id for p in products]
        for product_id in product_ids:
            for with_id in product_ids:
                # get the other products bought with each product
                if product_id != with_id:
                    # increment score for product purchased together
                    r.zincrby(self.get_product_key(product_id),
                              with_id,
                              amount=1)

```

این کلاس پیشنهادی است که به ما امکان می دهد خریدهای محصول را ذخیره کنیم و پیشنهادات محصول را برای یک محصول یا محصولات خاص بازیابی کنیم. متدهای `get_product_key()` شناسه یک شیء محصول را دریافت می کند و کلید Redis را برای مجموعه مرتب شده ای که محصولات مرتبط در آن ذخیره می شوند، می سازد، که به نظر می رسد محصول: `.with[id]`.

- متدهای `products_bought()` لیستی از اشیاء محصول را که با هم خریداری شده اند دریافت می کند (یعنی متعلق به همان سفارش). در این روش کارهای زیر را انجام می دهیم:
1. ما شناسه محصول را برای اشیاء محصول داده شده دریافت می کنیم.
 2. ما بیش از شناسه محصول تکرار می کنیم. برای هر شناسه، بیشتر از شناسه محصول تکرار می کنیم و همان محصول را رد می کنیم تا کالاهایی را که با هر محصول خریداری می شود، بدست آوریم.

3. ما برای هر محصول خریداری شده با استفاده از روش Redis () کلید محصول get_product_id را بدست می آوریم. برای یک محصول با شناسه 33 ، این روش محصول اصلی را باز می گرداند: buy_with_33:33. این کلید دسته بندی مرتب شده ای است که شامل شناسه محصولاتی است که به همراه این مجموعه خریداری شده اند.

4- نمره هر شناسه محصول موجود در مرتب سازی بر اساس 1 را افزایش می دهیم. امتیاز نشانگر زمانی است که کالای دیگری به همراه محصول داده شده خریداری شده است.

بنابراین ما روشی برای ذخیره و امتیاز دادن به محصولاتی که با هم خریداری شده اند ، داریم. اکنون به روشی برای بازیابی کالاهایی که برای یک لیست از محصولات داده شده با هم خریداری می شوند نیاز داریم. روش sug_products_for زیر را به کلاس توصیه کننده اضافه کنید:

```
def suggest_products_for(self, products, max_results=6):
    product_ids = [p.id for p in products]
    if len(products) == 1:
        # only 1 product
        suggestions = r.zrange(
            self.get_product_key(product_ids[0]),
            0, -1, desc=True)[:max_results]
    else:
        # generate a temporary key
        flat_ids = ''.join([str(id) for id in product_ids])
        tmp_key = 'tmp_{}'.format(flat_ids)
        # multiple products, combine scores of all products
        # store the resulting sorted set in a temporary key
        keys = [self.get_product_key(id) for id in product_ids]
        r.zunionstore(tmp_key, keys)
        # remove ids for the products the recommendation is for
        r.zrem(tmp_key, *product_ids)
        # get the product ids by their score, descendant sort
        suggestions = r.zrange(tmp_key, 0, -1,
                               desc=True)[:max_results]
        # remove the temporary key
        r.delete(tmp_key)
    suggested_products_ids = [int(id) for id in suggestions]

    # get suggested products and sort by order of appearance
    suggested_products =
list(Product.objects.filter(id__in=suggested_products_ids))
    suggested_products.sort(key=lambda x: suggested_products_ids.index(x.id))
    return suggested_products
```

متداول پارامترهای زیر را دریافت می کند: sug_products_for

: این لیستی از اشیاء محصول برای دریافت توصیه هایی است. این می تواند شامل یک یا چند محصول باشد.

: این یک عدد صحیح است که حداقل تعداد توصیه ها برای بازگشت را نشان می دهد.

در این روش اقدامات زیر را انجام می دهیم:

1. ما شناسه محصول را برای اشیاء محصول داده شده دریافت می کنیم.

2. اگر فقط به یک محصول داده شده است ، ما شناسه کالاهایی را که به همراه کالای داده شده خریداری شده اند ، بازیابی می کنیم و به تعداد کل دفعات خریداری شده با هم ، سفارش می دهیم. برای این کار از دستور Redis 'ZRANGE' استفاده می کنیم. ما تعداد نتایج را به عدد مشخص شده در ویژگی max_results (6 به طور پیش فرض) محدود می کنیم.

3. اگر بیش از یک محصول داده شود ، ما یک کلید موقتی Redis که با شناسه محصولات ساخته شده است ، تولید می کنیم.

4. ما برای همه موارد موجود در مجموعه مرتب شده هر یک از محصولات داده شده ، تمام نمرات را جمع و جمع می کنیم. این کار با استفاده از دستور Redis 'ZUNIONSTORE' انجام می شود. دستور ZUNIONSTORE اتحادیه مجموعه های مرتب شده را با کلیدهای داده شده انجام می دهد و جمع کل نمرات عناصر را در یک کلید جدید Redis ذخیره می کند. می توانید اطلاعات بیشتر در مورد این دستور را در <https://redis.io/commands/ZUNIONSTORE> مطالعه کنید. ما نمرات جمع شده را در کلید موقتی ذخیره می کنیم.

5- از آنجا که نمرات جمع می کنیم ، ممکن است همان کالاهایی را که توصیه می کنیم به دست آوریم. ما آنها را با استفاده از دستور ZREM از مجموعه مرتب شده تولید شده حذف می کنیم.

6. ما شناسه محصولات را از کلید موقتی که به وسیله نمره آنها با استفاده از دستور RANGE سفارش داده می شود ، بازیابی می کنیم. ما تعداد نتایج را به عدد مشخص شده در ویژگی max_results محدود می کنیم. سپس کلید موقتی را حذف می کنیم.

7. در آخر ، اشیاء Product را با شناسه مشخص دریافت می کنیم و محصولات را به همان ترتیب سفارش می دهیم.

برای اهداف عملی ، بیایید روشی را نیز برای پاک کردن توصیه ها اضافه کنیم. روش زیر را به کلاس توصیه کننده اضافه کنید:

```
def clear_purchases(self):
    for id in Product.objects.values_list('id', flat=True):
        r.delete(self.get_product_key(id))
```

بیایید موتور توصیه ما را امتحان کنیم. اطمینان حاصل کنید که چندین اشیاء Product را در دیتابیس登録 کرده و سرور Redis را با استفاده از دستور زیر از پوسته موجود در فهرست دایرکتوری Redis خود شروع کنید:

```
src/redis-server
```

پوسته دیگری را باز کنید و دستور زیر را برای باز کردن پوسته پایتون اجرا کنید:

```
python manage.py shell
```

اطمینان حاصل کنید که حداقل چهار محصول مختلف در بانک اطلاعاتی خود داشته باشد.

چهار محصول مختلف را با نام خود بازیابی کنید:

```
>>> from shop.models import Product
>>> black_tea = Product.objects.get(translations__name='Black tea')
>>> red_tea = Product.objects.get(translations__name='Red tea')
>>> green_tea = Product.objects.get(translations__name='Green tea')
>>> tea_powder = Product.objects.get(translations__name='Tea powder')
```

سپس ، برخی از خریدهای آزمایشی را به موتور توصیه اضافه کنید:

```
>>> from shop.recommender import Recommender  
>>> r = Recommender()  
>>> r.products_bought([black_tea, red_tea])  
>>> r.products_bought([black_tea, green_tea])  
>>> r.products_bought([red_tea, black_tea, tea_powder])  
>>> r.products_bought([green_tea, tea_powder])  
>>> r.products_bought([black_tea, tea_powder])  
>>> r.products_bought([red_tea, green_tea])
```

ما نمرات زیر را ذخیره کرده ایم:

```
black_tea:  red_tea (2), tea_powder (2), green_tea (1)  
red_tea:    black_tea (2), tea_powder (1), green_tea (1)  
green_tea:   black_tea (1), tea_powder (1), red_tea(1)  
tea_powder: black_tea (2), red_tea (1), green_tea (1)
```

باید برای بازیابی محصولات ترجمه شده زبانی را فعال کنیم و توصیه های محصول را برای خرید همراه با یک محصول واحد خاص دریافت کنیم:

```
>>> from django.utils.translation import activate  
>>> activate('en')  
>>> r.suggest_products_for([black_tea])  
[<Product: Tea powder>, <Product: Red tea>, <Product: Green tea>]  
>>> r.suggest_products_for([red_tea])  
[<Product: Black tea>, <Product: Tea powder>, <Product: Green tea>]  
>>> r.suggest_products_for([green_tea])  
[<Product: Black tea>, <Product: Tea powder>, <Product: Red tea>]  
>>> r.suggest_products_for([tea_powder])  
[<Product: Black tea>, <Product: Red tea>, <Product: Green tea>]
```

می بینید که سفارش محصولات پیشنهادی براساس امتیاز آنها است. باید توصیه هایی را برای چندین محصول با نمرات جمع بندی دریافت کیم:

```
>>> r.suggest_products_for([black_tea, red_tea])
[<Product: Tea powder>, <Product: Green tea>]
>>> r.suggest_products_for([green_tea, red_tea])
[<Product: Black tea>, <Product: Tea powder>]
>>> r.suggest_products_for([tea_powder, black_tea])
[<Product: Red tea>, <Product: Green tea>]
```

می بینید که ترتیب محصولات پیشنهادی با نمرات جمع شده مطابقت دارد. به عنوان مثال ، محصولاتی که برای red_tea و black_tea پیشنهاد شده اند ، چای_پودر (1 + 2) و green_tea (1 + 1) هستند.

ما تأیید کرده ایم که الگوریتم توصیه ما همانطور که انتظار می رود کار می کند. باید توصیه های مربوط به محصولات را در سایت خود نمایش دهیم.

پرونده views.py برنامه فروشگاه را ویرایش کنید. برای بازیابی حداقل چهار محصول توصیه شده در نمای Product_detail به شرح زیر عملکردی را اضافه کنید:

```
from .recommender import Recommender

def product_detail(request, id, slug):
    language = request.LANGUAGE_CODE
    product = get_object_or_404(Product,
                                id=id,
                                translations__language_code=language,
                                translations__slug=slug,
                                available=True)
    cart_product_form = CartAddProductForm()

    r = Recommender()
    recommended_products = r.suggest_products_for([product], 4)

    return render(request,
                  'shop/product/detail.html',
                  {'product': product,
                   'cart_product_form': cart_product_form,
                   'recommended_products': recommended_products})
```

قالب product.description فروشگاه را ویرایش کنید و کد زیر را پس از } } اضافه کنید:

linebreaks add add

```
{% if recommended_products %}
  <div class="recommendations">
    <h3>{% trans "People who bought this also bought" %}</h3>
    {% for p in recommended_products %}
      <div class="item">
        <a href="{{ p.get_absolute_url }}>
          
        </a>
        <p><a href="{{ p.get_absolute_url }}>{{ p.name }}</a></p>
      </div>
    {% endfor %}
  </div>
{% endif %}
```

سرور توسعه را اجرا کنید و <http://127.0.0.1:8000/en/> را در مرورگر خود باز کنید. برای دیدن جزئیات آن بر روی هر محصول کلیک کنید. باید ببینید محصولات پیشنهادی در زیر محصول همانطور که در تصویر زیر نشان داده شده است نمایش داده می شوند:



Tea powder

Tea

\$21.2

Quantity: 1

Add to cart

People who bought this also bought



Black tea



Red tea



Green tea

ما همچنین قصد داریم توصیه‌های محصول را در سبد خرید وارد کنیم.

این توصیه بر اساس محصولاتی است که کاربر به سبد خرید اضافه کرده است.

views.py را درون برنامه سبد خرید ویرایش کنید ، کلاس پیشنهاد دهنده را وارد کنید و نمای سبد خرید را ویرایش کنید تا به صورت زیر ظاهر شود:

```

from shop.recommender import Recommender

def cart_detail(request):
    cart = Cart(request)
    for item in cart:
        item['update_quantity_form'] = CartAddProductForm(
            initial={'quantity': item['quantity'],
                      'update': True})

    coupon_apply_form = CouponApplyForm()

    r = Recommender()
    cart_products = [item['product'] for item in cart]
    recommended_products = r.suggest_products_for(cart_products,
                                                    max_results=4)

return render(request,
              'cart/detail.html',
              {'cart': cart,
               'coupon_apply_form': coupon_apply_form,
               'recommended_products': recommended_products})

```

قالب سبد / detail.html برنامه سبد خرید را ویرایش کرده و آن را اضافه کنید

کد زیر را دقیقاً بعد از بروچسب HTML / جدول زیر دنبال کنید:

```

{% if recommended_products %}
<div class="recommendations cart">
    <h3>{% trans "People who bought this also bought" %}</h3>
    {% for p in recommended_products %}
        <div class="item">
            <a href="{{ p.get_absolute_url }}>
                
            </a>
            <p><a href="{{ p.get_absolute_url }}>{{ p.name }}</a></p>
        </div>
    {% endfor %}
</div>

```

```
{% endif %}
```

Your shopping cart

Image	Product	Quantity	Remove	Unit price	Price
	Tea powder	1 <input type="button" value="Update"/>	<input type="button" value="Remove"/>	\$21.2	\$21.2
	Green tea	1 <input type="button" value="Update"/>	<input type="button" value="Remove"/>	\$30	\$30
Total					\$51.20

People who bought this also bought



Black tea

Red tea

Apply a coupon:

Coupon:

تبریک می‌گوییم! شما یک موتور توصیه کاملی با استفاده از Django و Redis ساخته اید.

خلاصه

در این فصل شما با استفاده از جلسات ، یک سیستم کوپن ایجاد کرده اید. شما یاد گرفتید که چگونه بین المللی و محلی سازی کار می کنند. شما همچنین با استفاده از Redis موتور پیشنهادی ساخته اید.

در فصل بعد پروژه جدیدی را شروع می کنید. شما با استفاده از نماهای مبتنی بر کلاس ، یک پلت فرم یادگیری با Django ایجاد خواهید کرد و یک سیستم مدیریت محتوای سفارشی ایجاد خواهید کرد.

فصل دهم

ساختن یک آموزش الکترونیکی

CMS

در فصل قبل ، بین المللی سازی را به پروژه فروشگاه اینترنتی خود اضافه کردید. شما همچنان یک سیستم کوپن و یک موتور توصیه محصول ساخته اید. در این فصل یک پروژه جدید ایجاد خواهید کرد. شما یک بستر یادگیری الکترونیکی ایجاد خواهید کرد و یک سفارشی ایجاد می کنید

سیستم مدیریت محتوا (CMS).

در این فصل یاد می گیرید که چگونه:

لوازم جانبی را برای مدل های خود ایجاد کنید

از وراثت مدل استفاده کنید

زمینه های مدل دلخواه را ایجاد کنید

از نماها و ترکیب های مبتنی بر کلاس استفاده کنید

ساخت قالبها

مدیریت گروه ها و مجوزها

ایجاد CMS

تنظیم پروژه یادگیری الکترونیک

پروژه عملی نهایی ما یک بستر یادگیری الکترونیک خواهد بود. در این فصل ، ما می خواهیم یک CMS انعطاف پذیر بسازیم که به مریبان اجازه می دهد دوره هایی را ایجاد کنند و محتویات خود را مدیریت کنند.

ابتدا یک محیط مجازی برای پروژه جدید خود ایجاد کرده و با دستورات زیر آن را فعال کنید:

```
mkdir env  
virtualenv env/educa  
source env/educa/bin/activate
```

را در محیط مجازی خود با دستور زیر نصب کنید:

```
pip install Django==2.0.5
```

ما در پروژه خود قصد داریم بارگذاری تصویر را مدیریت کنیم ، بنابراین باید دستور زیر را نیز با دستور زیر نصب کنیم:

```
pip install Pillow
```

با استفاده از دستور زیر یک پروژه جدید ایجاد کنید:

```
django-admin startproject educa
```

وارد پوشه آموزشی جدید شوید و با استفاده از برنامه کاربردی جدید ایجاد کنید

دستورات زیر:

```
cd educa
django-admin startapp courses
```

پرونده settings.py را ویرایش کنید و دوره هایی را به تنظیمات INSTALLED_APPS اضافه کنید:

```
INSTALLED_APPS = [
    'courses.apps.CoursesConfig',
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
]
```

برنامه دوره ها اکنون برای این پروژه فعال است. باید مدل ها را برای دوره ها و مطالب درسی تعریف کنیم.

ساخت مدل های دوره

پلت فرم یادگیری الکترونیکی ما دوره های مختلفی را ارائه می دهد. هر دوره به تعداد قابل تنظیم مأذول ها تقسیم می شود و هر یک از مأذول ها شامل یک تعداد قابل تنظیم از محتوا است. محتوای مختلف وجود دارد: متن، پرونده، تصویر یا فیلم. مثال زیر نشان می دهد ساختار داده های کاتالوگ دوره ما چگونه خواهد بود:

```
Subject 1
Course 1
Module 1
    Content 1 (image)
    Content 2 (text)
Module 2
    Content 3 (text)
    Content 4 (file)
    Content 5 (video)
    ...
```

باید مدل های دوره را بسازیم. پرونده model.py برنامه دوره را ویرایش کنید و کد زیر را به آن اضافه کنید:

```
from django.db import models
from django.contrib.auth.models import User

class Subject(models.Model):
    title = models.CharField(max_length=200)
    slug = models.SlugField(max_length=200, unique=True)

    class Meta:
        ordering = ['title']

    def __str__(self):
        return self.title

class Course(models.Model):
    owner = models.ForeignKey(User,
```

```

        related_name='courses_created',
        on_delete=models.CASCADE)
subject = models.ForeignKey(Subject,
                           related_name='courses',
                           on_delete=models.CASCADE)
title = models.CharField(max_length=200)
slug = models.SlugField(max_length=200, unique=True)
overview = models.TextField()
created = models.DateTimeField(auto_now_add=True)

class Meta:
    ordering = ['-created']

def __str__(self):
    return self.title


class Module(models.Model):
    course = models.ForeignKey(Course,
                               related_name='modules',
                               on_delete=models.CASCADE)
    title = models.CharField(max_length=200)
    description = models.TextField(blank=True)

    def __str__(self):
        return self.title

```

این مدل‌های اولیه Subject، Course و Module هستند. زمینه‌های مدل دوره به شرح زیر است:

صاحب: مربی که این دوره را ایجاد کرده است.

موضوع: موضوعی که این دوره متعلق به آن است. خارجی

زمینه‌ای که به مدل Subject اشاره دارد.

عنوان: عنوان دوره.

مثل حلقه حرکت کردن: خزنده دوره. بعداً در URL ها استفاده می‌شود.

بررسی اجمالی: این یک ستون TextField است که شامل یک دوره کلی از دوره است.

ایجاد شده: تاریخ و زمان ایجاد دوره.

این کار هنگام ایجاد اشیاء جدید به دلیل auto_now_add = درست توسط جنگو تنظیم می شود.

هر دوره به چند مأژول تقسیم می شود. بنابراین ، مدل مأژول شامل یک فیلد ForeignKey است که به مدل Course اشاره می کند.

پوسته را باز کنید و دستور زیر را برای ایجاد مهاجرت اولیه برای این برنامه اجرا کنید:

```
python manage.py makemigrations
```

خروجی زیر را مشاهده خواهید کرد:

```
Migrations for 'courses':
  0001_initial.py:
    - Create model Course
    - Create model Module
    - Create model Subject
    - Add field subject to course
```

سپس دستور زیر را اجرا کنید تا کلیه مهاجرت ها به پایگاه داده اعمال شود:

```
python manage.py migrate
```

شما باید خروجی شامل همه مهاجرت های کاربردی ، از جمله جنگو را ببینید. خروجی شامل خط زیر خواهد بود:

```
Applying courses.0001_initial... OK
```

مدل های برنامه دوره های ما با پایگاه داده همگام سازی شده اند.

ثبت مدل ها در سایت اداره

باید مدل های دوره را به سایت مدیریت اضافه کنیم. پرونده admin.py را درون دایرکتوری برنامه دوره ها ویرایش کنید و کد زیر را به آن اضافه کنید:

```
from django.contrib import admin
from .models import Subject, Course, Module

@admin.register(Subject)
class SubjectAdmin(admin.ModelAdmin):
    list_display = ['title', 'slug']
    prepopulated_fields = {'slug': ('title',)}

class ModuleInline(admin.StackedInline):
    model = Module

@admin.register(Course)
class CourseAdmin(admin.ModelAdmin):
    list_display = ['title', 'subject', 'created']
    list_filter = ['created', 'subject']
    search_fields = ['title', 'overview']
    prepopulated_fields = {'slug': ('title',)}
    inlines = [ModuleInline]
```

مدل های مربوط به درخواست دوره اکنون در سایت مدیریت ثبت شده است. به یاد داشته باشید ، ما برای ثبت مدل ها در سایت اداره از دکوراتور @ admin.register () استفاده می کنیم.

استفاده از لوازم جانبی برای تهیه داده های اولیه برای مدل ها

بعضی اوقات ممکن است بخواهید داده های خود را با داده های هارد کدگذاری شده از قبل جمع کنید. این کار مفید است که بجای اضافه کردن دستی به آن ، به طور خودکار داده های اولیه را در مجموعه پروژه قرار دهید. Django با روشی ساده برای بارگیری و ریختن داده از پایگاه داده در پرونده هایی که به آنها فیکسچر گفته می شود ، ارائه می شود.

جنگو از قالب های JSON، XML یا YAML پشتیبانی می کند. ما در حال ساختن یک فیکسچر هستیم که شامل چندین موضوع اولیه Subject برای پروژه ما باشد.

ابتدا با استفاده از دستور زیر یک superuser ایجاد کنید:

```
python manage.py createsuperuser
```

سپس سرور توسعه را با استفاده از دستور زیر اجرا کنید:

/ را در مرورگر خود باز کنید. چندین موضوع را با استفاده از سایت مدیریت ایجاد کنید. صفحه نمایش لیست باید به شرح زیر باشد:

Django administration

WELCOME, ADMIN | VIEW SITE / CHANGE PASSWORD / LOG OUT

Home > Courses > Subjects

Select subject to change

ADD SUBJECT +

Action: ----- Go 0 of 4 selected

TITLE

SLUG

Mathematics

mathematics

Music

music

Physics

physics

Programming

programming

4 subjects

دستور زیر را از پوسته اجرا کنید:

```
python manage.py dumpdata courses --indent=2
```

خروجی مشابه موارد زیر را مشاهده خواهید کرد:

```
[  
{  
    "model": "courses.subject",  
    "pk": 1,  
    "fields": {  
        "title": "Mathematics",  
        "slug": "mathematics"  
    }  
},  
{  
    "model": "courses.subject",  
    "pk": 2,  
    "fields": {  
        "title": "Music",  
        "slug": "music"  
    }  
},  
{  
    "model": "courses.subject",  
    "pk": 3,  
    "fields": {  
        "title": "Physics",  
        "slug": "physics"  
    }  
},  
{  
    "model": "courses.subject",  
    "pk": 4,  
    "fields": {  
        "title": "Programming",  
        "slug": "programming"  
    }  
}  
]
```

دستور dumpdata داده ها را از پایگاه داده به خروجی استاندارد رها می کند، که به طور پیش فرض در قالب JSON سریالی می شود. ساختار داده به دست آمده شامل اطلاعاتی در مورد مدل و زمینه های آن برای Django است تا بتواند آن را در پایگاه داده بارگذاری کند.

شما می توانید با تهیه نام برنامه به فرمان یا مشخص کردن مدل های واحد برای خروجی داده با استفاده از فرمت app.Model ، خروجی را به مدل های یک برنامه محدود کنید. همچنین می توانید قالب را با استفاده از پرچم --format مشخص کنید. به طور پیش فرض ، داده های سریالی شده را به خروجی استاندارد منتقل می کند. با این حال ، شما می توانید یک

فایل خروجی را با استفاده از پرچم output-- به شما امکان می دهد تا تورفتگی را مشخص کنید.
برای اطلاعات بیشتر در مورد پارامترهای Python management.py dumpdata - help، dumpdata را اجرا کنید.

با استفاده از دستورات زیر این دامپینگ را در یک پرونده فیکسچرها در یک لیست فیکسچرها / فهرست راهنمای ذخیره کنید:

```
mkdir courses/fixtures  
python manage.py dumpdata courses --indent=2 --  
output=courses/fixtures/subjects.json
```

سرور توسعه را اجرا کنید و از سایت مدیریت استفاده کنید تا موضوعاتی که ایجاد کرده اید حذف شود. سپس ، با استفاده از دستور زیر ، فیکسچر را در دیتابیس بارگذاری کنید:

```
python manage.py loaddata subjects.json
```

کلیه اشیاء Subject موجود در بخش در دیتابیس بارگذاری می شود.

به طور پیش فرض ، Django به دنبال فایل های موجود در فهرست / شاخه های هر برنامه است ، اما می توانید مسیر کامل مربوط به پرونده فیکسچر را برای فرمان loaddata مشخص کنید. همچنین می توانید از تنظیمات FIXTURE_DIRS استفاده کنید تا به دایرکتوری های اضافی Django بگویید تا به دنبال لوازم جانبی باشند.

می توانید در مورد چگونگی استفاده از لوازم جانبی برای آزمایش در استفاده کنید. <https://docs.djangoproject.com/en/2.0/topics/testing/tools/#fixture-loading>

اگر می خواهید وسایل مربوط به انتقال مدل را بارگیری کنید ، نگاهی به اسناد جنگو در مورد انتقال داده ها بیندازید. می توانید مستندات انتقال داده را در

<https://docs.djangoproject.com/en/2.0/topics/migrations/#data-migrations>

ایجاد مدل های برای مطالب متنوع

ما قصد داریم انواع مختلفی از مطالب را به ماژول های دوره مانند متن ، تصاویر ، فایل ها و فیلم ها اضافه کنیم. ما به یک مدل داده همه کاره نیاز داریم که به ما امکان ذخیره محتواهای متنوع را می دهد. در فصل 6 ، ردیابی عملکردهای کاربر ، شما به راحتی از استفاده از روابط عمومی برای ایجاد کلیدهای خارجی یادگرفته اید که می توانند به اشیاء هر مدل اشاره کنند. ما قصد داریم یک مدل محتوا ایجاد کنیم که نمایانگر محتویات ماژول ها باشد و یک رابطه عمومی را برای ارتباط هر نوع محتوا تعریف کنیم.

پرونده model.py برنامه دوره را ویرایش کنید و واردات زیر را اضافه کنید:

```
from django.contrib.contenttypes.models import ContentType
from django.contrib.contenttypes.fields import GenericForeignKey
```

سپس کد زیر را در انتهای پرونده اضافه کنید:

```
class Content(models.Model):
    module = models.ForeignKey(Module,
                               related_name='contents',
                               on_delete=models.CASCADE)
    content_type = models.ForeignKey(ContentType,
                                    on_delete=models.CASCADE)
    object_id = models.PositiveIntegerField()
    item = GenericForeignKey('content_type', 'object_id')
```

این مدل محتوا است. ماژول شامل چندین محتوا است ، بنابراین ما یک قسمت ForeignKey را به مدل ماژول تعریف می کنیم. ما همچنین از روابط عمومی با اشیاء وابسته از مدلهای مختلفی که انواع مختلفی از محتوا را نشان می دهند ، رابطه عام

تنظیم کردیم. به پاد داشته باشید که برای برقاری یک رابطه عمومی به سه زمینه مختلف نیاز داریم. در مدل محتوای ما ، این موارد است:

ContentType: یک قسمت ForeignKey به مدل content_type

object_id: PositiveIntegerField است برای ذخیره کلید اصلی شی مرتبط

مورد: یک زمینه GenericForeignKey با ترکیب دو قسمت قبلی با یک موضوع مرتبط مرتبط می شود

فقط قسمت های object_id و content_type دارای یک ستون مربوطه هستند

در جدول بانک اطلاعاتی این مدل. قسمت مورد شما اجازه می دهد تا شیء مربوط را مستقیماً بازیابی یا تنظیم کنید ، و قابلیت آن در بالای دو قسمت دیگر ساخته شده است.

ما قصد داریم از یک مدل متفاوت برای هر نوع محتوا استفاده کنیم. مدل‌های محتوای ما چندین قسمت مشترک دارند ، اما در داده های واقعی که می توانند ذخیره کنند متفاوت خواهند بود.

استفاده از ارث بری در مدل

جنگو از ارث مدل پشتیبانی می کند. این کار به روشنی مشابه با میراث طبقاتی استاندارد در پایتون کار می کند. Django سه گزینه زیر را برای استفاده از ارث مدل ارائه می دهد:

مدل‌های انتزاعی: وقتی می خواهید اطلاعات متداول را در چندین مدل قرار دهید مفید است. هیچ جدول بانک اطلاعاتی برای مدل انتزاعی ایجاد نشده است.

وراثت مدل چند جدول: هنگامی که هر مدل در سلسله مراتب به خودی خود یک مدل کامل در نظر گرفته شود قابل اجرا است. یک جدول بانک اطلاعاتی برای هر مدل ایجاد می شود.

مدل‌های پروکسی: هنگامی که شما نیاز به تغییر رفتار یک مدل دارید ، مثلاً با شامل روش های اضافی ، تغییر مدیر پیش فرض یا استفاده از گزینه های متأخّل ، مفید است. هیچ جدول بانک اطلاعاتی برای مدل های پراکسی ایجاد نشده است.

مدلهای انتزاعی

یک مدل انتزاعی یک کلاس پایه است که در آن شما می توانید زمینه های را که می خواهید در همه مدل های کودک درج کنید ، تعریف کنید. جنگو هیچ جدول بانک اطلاعاتی برای مدل های انتزاعی ایجاد نمی کند. برای هر مدل کودک یک جدول بانک اطلاعاتی ایجاد می شود ، از جمله زمینه هایی که از کلاس انتزاعی به دست می آید و موارد مشخص شده در مدل کودک.

برای علامت گذاری یک مدل به عنوان انتزاعی ، باید انتزاعی = True را در کلاس متا آن قرار دهید. جنگو تشخیص خواهد داد که این یک مدل انتزاعی است و جدول پایگاه داده برای آن ایجاد نمی کند. برای ایجاد مدل های کودک ، فقط باید مدل انتزاعی را طبقه بندی کنید.

مثال زیر یک مدل محتوای انتزاعی و یک متن متن کودک را نشان می دهد:

```
from django.db import models

class BaseContent(models.Model):
    title = models.CharField(max_length=100)
    created = models.DateTimeField(auto_now_add=True)

    class Meta:
        abstract = True

class Text(BaseContent):
    body = models.TextField()
```

در این حالت ، جنگو یک جدول فقط برای مدل متن ایجاد می کند ، از جمله عنوان ، ایجاد شده و قسمت های بدنه.

وراثت مدل چند جدول

در وراثت چند جدول ، هر مدل با یک جدول پایگاه داده مطابقت دارد. Django یک زمینه OneToOneField را برای ایجاد رابطه در مدل کودک با والدینش ایجاد می کند.

برای استفاده از وراثت چند جدول ، شما باید یک مدل موجود را طبقه بندی کنید. جنگو یک جدول بانک اطلاعاتی برای مدل اصلی و مدل فرعی ایجاد می کند. مثال زیر میراث چند جدول را نشان می دهد:

```
from django.db import models

class BaseContent(models.Model):
    title = models.CharField(max_length=100)
    created = models.DateTimeField(auto_now_add=True)

class Text(BaseContent):
    body = models.TextField()
```

جنگو شامل یک زمینه OneToOneField است که به صورت خودکار در مدل متن ایجاد می شود و برای هر مدل یک جدول پایگاه داده ایجاد می کند.

مدل های پروکسی

مدل های پروکسی برای تغییر رفتار یک مدل به عنوان مثال با استفاده از روش های اضافی یا گزینه های متأ مختلف استفاده می شود. هر دو مدل در جدول بانک اطلاعاتی مدل اصلی کار می کنند. برای ایجاد یک مدل پروکسی ، proxy=True را به کلاس متأ مدل اضافه کنید.

مثال زیر نحوه ایجاد یک مدل پروکسی را نشان می دهد:

```
from django.db import models
from django.utils import timezone

class BaseContent(models.Model):
    title = models.CharField(max_length=100)
    created = models.DateTimeField(auto_now_add=True)

class OrderedContent(BaseContent):
    class Meta:
        proxy = True
        ordering = ['created']

    def created_delta(self):
        return timezone.now() - self.created
```

در اینجا ، ما یک مدل OrderedContent را تعریف می کنیم که یک مدل پروکسی برای مدل Content است. این مدل یک سفارش پیش فرض برای QuerySets و یک روش اضافی ایجاد شده `__delta__(self)` ایجاد می کند. هر دو مدل ، Content و OrderedContent در همان جدول پایگاه داده عمل می کنند و اشیاء از طریق هر مدل از طریق ORM قابل دسترسی هستند.

ایجاد مدل های محتوا

مدل محتوا از برنامه دوره های ما حاوی یک رابطه عمومی برای مرتبط کردن انواع مختلف محتوا با آن است. ما برای هر نوع محتوا یک مدل متفاوت ایجاد خواهیم کرد. برای ذخیره داده های سفارشی ، کلیه مدل های محتوا دارای فیلد های مختلف در زمینه های مشترک و اضافی خواهند بود. ما در حال ایجاد یک مدل انتزاعی هستیم که زمینه های مشترک برای همه مدل های محتوا را فراهم می کند.

برونده `model.py` برنامه دوره را ویرایش کنید و کد زیر را به آن اضافه کنید:

```

class ItemBase(models.Model):
    owner = models.ForeignKey(User,
                              related_name='%(class)s_related',
                              on_delete=models.CASCADE)
    title = models.CharField(max_length=250)
    created = models.DateTimeField(auto_now_add=True)
    updated = models.DateTimeField(auto_now=True)

    class Meta:
        abstract = True

    def __str__(self):
        return self.title

class Text(ItemBase):
    content = models.TextField()

class File(ItemBase):
    file = models.FileField(upload_to='files')

class Image(ItemBase):
    file = models.FileField(upload_to='images')

class Video(ItemBase):
    url = models.URLField()

```

در این کد یک مدل انتزاعی با نام ArticleBase تعریف می‌کنیم. بنابراین، ما انتزاعی = True را در کلاس Meta قرار داده‌ایم. در این مدل زمینه‌های مالک، عنوان، ایجاد شده و به روز شده را تعریف می‌کنیم. این زمینه‌های مشترک برای همه نوع محتوا استفاده می‌شود. قسمت مالکیت به ما امکان می‌دهد که کدام کاربر محتوای را ایجاد کرده است. از آنجایی که این قسمت در یک کلاس انتزاعی تعریف شده است، برای هر مدل فرعی نیاز به نام‌های مختلف مرتبط داریم. Django به ما امکان می‌دهد یک مکان نگهدارنده برای نام کلاس مدل را در خصوصیت مربوط به_نام به عنوان %s (class) s مشخص کنیم. با این کار، برای هر مدل کودک به طور خودکار تولید می‌شود. از آنجا که ما از "%(class)s_related" استفاده می‌کنیم، رابطه معکوس برای مدل‌های کودک به ترتیب متناظر خواهد بود.

ما چهار مدل محتوا متفاوت تعریف کرده ایم که از مدل انتزاعی ArticleBase به ارث می‌برند. موارد زیر به شرح زیر است:

متن: برای ذخیره محتوای متن

پرونده: برای ذخیره فایلها ، مانند PDF

تصویر: برای ذخیره فایل های تصویری

ویدیو: برای ذخیره فیلم؛ ما برای تهیه یک URL از یک URL برای تهیه یک URL ویدیویی استفاده می کنیم

هر مدل کودک شامل زمینه های است که در کلاس ArticleBase تعریف شده است

علاوه بر زمینه های خاص خود یک جدول دیتابیس به ترتیب برای مدل های متن ، فایل ، تصویر و فیلم ایجاد می شود. از آنجا که این یک مدل انتزاعی است ، هیچ جدول پایگاه داده مرتبط با مدل ArticleBase وجود نخواهد داشت.

مدل محتوای را که قبلاً ایجاد کرده اید ویرایش کنید و قسمت content_type آن را به شرح زیر تغییر دهید:

```
content_type = models.ForeignKey(ContentType,  
                                on_delete=models.CASCADE,  
                                limit_choices_to={'model__in':(  
                                    'text',
```

```
'video',  
'image',  
'file'))
```

ما یک آرگومان limit_choices_to اضافه می کنیم تا اشیاء ContentType را که می تواند برای رابطه عمومی استفاده شود محدود کنیم. ما از جستجوی قسمت model_in برای فیلتر کردن پرس و جو به اشیاء ContentType با یک ویژگی مدل استفاده می کنیم که "متن" ، "ویدئو" ، "تصویر" یا "پرونده" است.

باید یک مهاجرت ایجاد کنیم تا مدل‌های جدیدی را که اضافه کرده ایم نیز بگنجانیم.

دستور زیر را از خط فرمان اجرا کنید:

```
python manage.py makemigrations
```

خروجی زیر را مشاهده خواهید کرد:

```
Migrations for 'courses':
  courses/migrations/0002_content_file_image_text_video.py
    - Create model Content
    - Create model File
    - Create model Image
    - Create model Text
    - Create model Video
```

سپس دستور زیر را اعمال کنید تا مهاجرت جدید اعمال شود:

```
python manage.py migrate
```

خروجی که می بینید باید با خطوط زیر پایان یابد:

```
Applying courses.0002_content_file_image_text_video... OK
```

ما مدل‌هایی را ایجاد کرده ایم که برای اضافه کردن محتوای متنوع به ماژول‌های دوره مناسب هستند. با این حال ، هنوز چیزی در مدل‌های ما وجود ندارد. ماژول‌ها و محتویات دوره باید از دستور خاصی پیروی کنند. ما به فیلدی احتیاج داریم که به ما اجازه دهد به راحتی سفارش دهیم

.

ایجاد زمینه‌های مدل‌های سفارشی

Django با مجموعه کاملی از زمینه‌های مدل ارائه شده است که می‌توانید برای ساخت مدل‌های خود استفاده کنید. با این حال ، شما همچنین می‌توانید زمینه‌های مدل خود را برای ذخیره داده‌های سفارشی یا تغییر رفتار زمینه‌های موجود ایجاد کنید.

ما به فیلدی احتیاج داریم که به ما امکان می‌دهد نظمی را برای اشیاء تعریف کنیم. یک روش آسان برای تعیین سفارش برای اشیاء با استفاده از فیلدهای موجود جنگو ، اضافه کردن یک PositiveIntegerField به مدل‌های شما است. با استفاده از اعداد صحیح می‌توان ترتیب اشیاء را به راحتی مشخص کرد. ما می‌توانیم یک فیلد سفارش سفارشی ایجاد کنیم که از PositiveIntegerField به ارث برسد و رفتار اضافی را فراهم کند.

دو کارکرد مرتبط وجود دارد که ما آنها را در قسمت سفارش خود ایجاد خواهیم کرد:

به طور خودکار مقدار سفارش را هنگامی که سفارش خاصی ارائه نشده است اختصاص دهید: هنگام ذخیره یک شیء جدید و بدون سفارش خاص ، قسمت ما باید به طور خودکار عددی را که بعد از آخرین شی دستور موجود موجود است ، اختصاص دهیم. اگر به ترتیب دو شیء به ترتیب 1 و 2 وجود داشته باشد ، هنگام ذخیره یک شیء سوم ، اگر سفارش خاصی ارائه نشده باشد ، باید به طور خودکار دستور 3 را به آن اختصاص دهیم.

ترتیب اشیاء با توجه به زمینه های دیگر: مازول های دوره با توجه به دوره متعلق به آنها و محتویات مازول با توجه به مازول مورد نظر آنها سفارش داده می شوند.

یک پرونده جدید Field.py را درون فهرست برنامه های کاربردی دوره ها ایجاد کنید و کد زیر را به آن اضافه کنید:

این OrderField سفارشی ما است. این قسمت از قسمت PositiveIntegerField ارائه شده توسط Django ارث می برد. قسمت OrderField ما از پارامتر `for_fields` اختیاری استفاده می کند که به ما امکان می دهد فیلدهای مورد نظر را با توجه به محاسبه نشان دهیم.

روش `pre_save()` از قسمت مثبت PositiveIntegerField عبور می کند ، که قبل از ذخیره کردن فیلد در پایگاه داده اجرا می شود. در این روش اقدامات زیر را انجام می دهیم:

1. ما بررسی می کنیم که آیا مقداری قبل از این فیلد در نمونه مدل وجود دارد یا خیر. ما از `self.attname` استفاده می کنیم ، که همان صفتی است که در مدل به فیلد داده شده است. اگر مقدار ویژگی از `None` متفاوت باشد ، ترتیب مورد نظر را باید به شرح زیر محاسبه کنیم:

1. ما `QuerySet` را برای بازیابی همه اشیاء برای مدل زمینه ایجاد می کنیم. ما با دستیابی به `self.model` کلاس را که به آن تعلق دارد ، بازیابی می کنیم.

2. `QuerySet` را با مقدار فعلی فیلدهای مدل که در پارامتر `for_fields` این فیلد تعریف شده اند ، فیلتر می کنیم ، در صورت وجود. با این کار ، ترتیب را با توجه به زمینه های داده شده محاسبه می کنیم.

3. ما شیء را با بالاترین مرتبه با `(self.attname, qs.latest(last_item))` از پایگاه داده بازیابی می کنیم. اگر هیچ جسم یافت نشد ، فرض می کنیم این شیء اولین مورد است و دستور 0 را به آن اختصاص می دهیم.

4- در صورت یافتن یک شی ، 1 مورد را به بالاترین مرتبه یافت شده اضافه می کنیم.

5- ترتیب محاسبه شده را به عنوان مثال در نمونه مدل با استفاده از `setattr()` با مقدار فیلد اختصاص داده و آن را برمی گردانیم.

2. اگر نمونه مدل برای زمینه فعلی یک مقدار دارد ، ما کاری انجام نمی دهیم. وقتی فیلدهای مدل دلخواه را ایجاد می کنید ، آنها را عمومی کنید. از داده های کدگذاری که به یک مدل یا فیلد خاص بستگی دارد ، خودداری کنید. زمینه شما باید در هر مدلی کار کند.

می توانید اطلاعات بیشتر در مورد نوشتمن زمینه های مدل های سفارشی را در <https://docs.djangoproject.com/en/2.0/howto/custom-model-fields> کسب کنید.

افزودن سفارش به اشیاء مازول و محتوا

بیایید زمینه جدید را به مدل‌های خود اضافه کنیم. پرونده model.py از برنامه دوره را ویرایش کرده و کلاس OrderField و یک فیلد را به مدل Module به شرح زیر وارد کنید:

```
from .fields import OrderField

class Module(models.Model):
    # ...
    order = OrderField(blank=True, for_fields=['course'])
```

ما ترتیب رشته جدید را نامگذاری می‌کنیم و با تنظیم `for_fields = ['course']` سفارش را با توجه به دوره محاسبه می‌کنیم. این بدان معنی است که سفارش برای یک مازول جدید اضافه می‌شود ۱ به آخرین مازول از همان شیء Course. اکنون می‌توانیم روش `__str__()` مدل مازول را ویرایش کنید تا ترتیب آن به شرح زیر باشد:

```
class Module(models.Model):
    # ...
    def __str__(self):
        return '{}. {}'.format(self.order, self.title)
```

محتویات مازول نیز باید دستور خاصی را دنبال کند. فیلد OrderField را به صورت زیر به مدل Content اضافه کنید:

```
class Content(models.Model):
    # ...
    order = OrderField(blank=True, for_fields=['module'])
```

این بار مشخص می کنیم که سفارش با توجه به قسمت مأذول محاسبه می شود. در آخر ، اجازه دهید یک سفارش پیش فرض برای هر دو مدل اضافه کنیم. کلاس متای زیر را به مدل های مأذول و محتوا اضافه کنید:

```
class Module(models.Model):
    # ...
    class Meta:
        ordering = ['order']

class Content(models.Model):
    # ...
    class Meta:
        ordering = ['order']
```

مدل های مأذول و محتوا اکنون باید به شرح زیر باشد:

```

class Module(models.Model):
    course = models.ForeignKey(Course,
                               related_name='modules',
                               on_delete=models.CASCADE)
    title = models.CharField(max_length=200)
    description = models.TextField(blank=True)
    order = OrderField(blank=True, for_fields=['course'])

    class Meta:
        ordering = ['order']

    def __str__(self):
        return '{}. {}'.format(self.order, self.title)

class Content(models.Model):
    module = models.ForeignKey(Module,
                               related_name='contents',
                               on_delete=models.CASCADE)
    content_type = models.ForeignKey(ContentType,
                                     on_delete=models.CASCADE,
                                     limit_choices_to={'model__in':(
                                         'text',
                                         'video',
                                         'image',
                                         'file')})
    object_id = models.PositiveIntegerField()
    item = GenericForeignKey('content_type', 'object_id')
    order = OrderField(blank=True, for_fields=['module'])

    class Meta:
        ordering = ['order']

```

بیایید یک مدل مهاجرت جدید ایجاد کنیم که منعکس کننده قسمت های سفارش جدید باشد. پوسته را باز کرده و دستور زیر را اجرا کنید:

```
python manage.py makemigrations courses
```

خروجی زیر را مشاهده خواهید کرد:

```
You are trying to add a non-nullable field 'order' to content without a
default; we can't do that (the database needs something to populate existing
rows).
Please select a fix:
1) Provide a one-off default now (will be set on all existing rows with a
null value for this column)
2) Quit, and let me add a default in models.py
Select an option:
```

به ما می گوید که ما باید یک مقدار پیش فرض برای فیلد سفارش جدید برای سطرهای موجود در دیتابیس ارائه دهیم. اگر این فیلد nullable باشد ، مقادیر تهی را می پذیرید و جنگو به جای درخواست یک مقدار پیش فرض ، مهاجرت را به طور خودکار ایجاد می کند. ما می توانیم یک مقدار پیش فرض را تعیین کنیم یا مهاجرت را لغو کنیم و قبل از ایجاد مهاجرت ، یک ویژگی پیش فرض را به قسمت ترتیب در پرونده model.py اضافه کنیم.

1 را وارد کنید و Enter را فشار دهید تا یک مقدار پیش فرض برای سوابق موجود فراهم شود. خروجی زیر را مشاهده خواهید کرد:

```
Please enter the default value now, as valid Python
The datetime and django.utils.timezone modules are available, so you can do
e.g. timezone.now
Type 'exit' to exit this prompt
>>>
```

0 را وارد کنید تا این مقدار پیش فرض برای سوابق موجود باشد و Enter را فشار دهید. همچنین جنگو از شما می خواهد یک مقدار پیش فرض برای مدل ماثول نیز بخواهد. اولین گزینه را انتخاب کرده و 0 را به عنوان مقدار پیش فرض دوباره وارد کنید. سرانجام ، خروجی مشابه با مورد زیر را مشاهده خواهید کرد:

```
Migrations for 'courses':  
  courses/migrations/0003_auto_20180326_0704.py  
    - Change Meta options on content  
    - Change Meta options on module  
    - Add field order to content  
    - Add field order to module
```

سپس مهاجرت های جدید را با دستور زیر اعمال کنید:

خروجی فرمان به شما اطلاع می دهد که مهاجرت با موفقیت انجام شد ، به شرح زیر:

```
Applying courses.0003_auto_20180326_0704... OK
```

باید زمینه جدید خود را آزمایش کنیم. پوسته را با دستور زیر باز کنید:

```
python manage.py shell
```

دوره جدیدی را به شرح زیر ایجاد کنید:

```
>>> from django.contrib.auth.models import User  
>>> from courses.models import Subject, Course, Module  
>>> user = User.objects.last()  
>>> subject = Subject.objects.last()  
>>> c1 = Course.objects.create(subject=subject, owner=user, title='Course 1',  
slug='course1')
```

ما یک دوره در بانک اطلاعاتی ایجاد کرده ایم. حال باید مازول ها را به دوره اضافه کنیم و ببینیم که چگونه ترتیب آنها به طور خودکار محاسبه می شود. ما یک مازول اولیه ایجاد می کنیم و ترتیب آن را بررسی می کنیم:

```
>>> m1 = Module.objects.create(course=c1, title='Module 1')
>>> m1.order
0
```

مقدار خود را برابر 0 قرار می دهد ، زیرا این اولین شیء ماژول است که برای دوره معین ایجاد شده است. اکنون ، ماژول دوم را برای همان دوره ایجاد می کنیم:

```
>>> m2 = Module.objects.create(course=c1, title='Module 2')
>>> m2.order
1
```

مقدار سفارش بعدی را با اضافه کردن 1 به بالاترین مرتبه برای اشیاء موجود محاسبه می کند. بیایید یک ماژول سوم ایجاد کنیم و یک سفارش خاص را مجبور کنیم:

```
>>> m3 = Module.objects.create(course=c1, title='Module 3', order=5)
>>> m3.order
5
```

اگر یک سفارش سفارشی را تعیین کنیم ، قسمت OrderField دخالت نمی کند و از مقدار داده شده به سفارش استفاده می شود.

بیایید یک ماژول چهارم اضافه کنیم:

```
>>> m4 = Module.objects.create(course=c1, title='Module 4')
>>> m4.order
6
```

سفارش این ماژول بطور خودکار تنظیم شده است. قسمت OrderField ما تضمین نمی کند که همه مقادیر مرتب هستند.

با این وجود ، به مقادیر سفارش موجود احترام می کنار و همیشه سفارش بعدی را براساس بالاترین سفارش موجود اختصاص می دهد.

باید یک دوره دوم ایجاد کنیم و یک مژول به آن اضافه کنیم:

```
>>> c2 = Course.objects.create(subject=subject, title='Course 2',  
slug='course2', owner=user)  
>>> m5 = Module.objects.create(course=c2, title='Module 1')  
>>> m5.order  
0
```

برای محاسبه سفارش مژول جدید ، این فیلد فقط مژول های موجود را در نظر می گیرد که به همان دوره تعلق دارند. از آنجاکه این مژول اول دوره دوم است ، نتیجه حاصله 0. است. زیرا ما برای قسمت مربوط به مدل مژول for_fields = ["course"] را مشخص کردیم.

تبریک می گوییم! شما با موفقیت اولین قسمت مدل دلخواه خود را ایجاد کرده اید.

ایجاد CMS

اکنون که ما یک مدل داده همه کاره را ایجاد کرده ایم ، قصد داریم CMS را بسازیم. به مریان این امکان را می دهد که دوره هایی را ایجاد کرده و مطالب خود را مدیریت کنند. ما باید عملکردهای زیر را ارائه دهیم:

وارد CMS شوید

دوره های ایجاد شده توسط مریان را لیست کنید

دوره ها را ایجاد ، ویرایش و حذف کنید

مژول ها را به یک دوره اضافه کنید و آنها را دوباره ترتیب دهید

انواع مختلفی از محتوا را به هر مژول اضافه کرده و ترتیب دهید

اضافه کردن یک سیستم تأیید اعتبار

ما می خواهیم از چارچوب تأیید هویت Django در پلتفرم خود استفاده کنیم. مریبان و دانشجویان نیز نمونه هایی از الگوی کاربر Django هستند ، بنابراین آنها قادر خواهند بود با استفاده از نماهای احراز هویت django.contrib.auth وارد سایت شوند.

پرونده اصلی urls.py پروژه آموزشی را ویرایش کنید و نمای ورودی و خروج از چارچوب احراز هویت Django را درج کنید:

```
from django.contrib import admin
from django.urls import path
from django.contrib.auth import views as auth_views

urlpatterns = [
    path('accounts/login/', auth_views.LoginView.as_view(), name='login'),
    path('accounts/logout/', auth_views.LogoutView.as_view(), name='logout'),
    path('admin//', admin.site.urls),
]
```

اضافه کردن یک سیستم تأیید اعتبار

ما می خواهیم از چارچوب تأیید هویت Django در پلتفرم خود استفاده کنیم. مریبان و دانشجویان نیز نمونه هایی از الگوی کاربر Django هستند ، بنابراین آنها قادر خواهند بود با استفاده از نماهای احراز هویت django.contrib.auth وارد سایت شوند.

پرونده اصلی urls.py پروژه آموزشی را ویرایش کنید و نمای ورودی و خروج از چارچوب احراز هویت Django را درج کنید:

```
from django.contrib import admin
from django.urls import path
from django.contrib.auth import views as auth_views

urlpatterns = [
    path('accounts/login/', auth_views.LoginView.as_view(), name='login'),
    path('accounts/logout/', auth_views.LogoutView.as_view(), name='logout'),
    path('admin/', admin.site.urls),
]
```

ایجاد الگوهای تأیید اعتبار

ساختار پرونده زیر را در فهرست برنامه کاربردی دوره ها ایجاد کنید:

```
templates/
base.html
registration/
login.html
logged_out.html
```

قبل از ساختن الگوهای تأیید اعتبار ، باید الگوی پایه را برای پروژه خود آماده کنیم. پرونده قالب base.html را ویرایش کنید و محتوای زیر را به آن اضافه کنید:

```

{% load staticfiles %}
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <title>{% block title %}Educa{% endblock %}</title>
    <link href="{% static "css/base.css" %}" rel="stylesheet">
</head>
<body>
    <div id="header">
        <a href="/" class="logo">Educa</a>
        <ul class="menu">
            {% if request.user.is_authenticated %}
                <li><a href="{% url "logout" %}">Sign out</a></li>
            {% else %}
                <li><a href="{% url "login" %}">Sign in</a></li>
            {% endif %}
        </ul>
    </div>
    <div id="content">
        {% block content %}
        {% endblock %}
    </div>

    <script src="https://ajax.googleapis.com/ajax/libs/jquery/
3.3.1/jquery.min.js"></script>
<script>
    $(document).ready(function() {
        {% block domready %}
        {% endblock %}
    });
</script>
</body>
</html>

```

این الگوی پایه ای است که توسط بقیه قالب ها گسترش می یابد. در این الگو بلوک های زیر را تعریف می کنیم:

عنوان: بلوک قالب های دیگر برای اضافه کردن عنوان سفارشی برای هر صفحه.

محتوا: بلوک اصلی برای محتوا. کلیه قالب هایی که الگوی پایه را گسترش می دهند باید به این بخش محتوای اضافه کنند.

در حال حاضر: در داخل تابع `jQuery().re()` واقع شده است.

این امکان را به ما می دهد تا وقتی بار DOM به پایان رسید بار اجرای کد را انجام دهیم.

سبک های CSS استفاده شده در این الگو در استاتیک / دایرکتوری برنامه دوره ها ، در کدی که همراه با این فصل قرار دارد ، قرار دارند. استاتیک / فهرست را در همان فهرست کپی کنید

پروژه خود را برای استفاده از آنها

الگوی ثبت نام / login.html را ویرایش کنید و کد زیر را به آن اضافه کنید:

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

{% block title %}Log-in{% endblock %}

{% block content %}
```

```
<h1>Log-in</h1>
<div class="module">
    {% if form.errors %}
        <p>Your username and password didn't match. Please try again.</p>
    {% else %}
        <p>Please, use the following form to log-in:</p>
    {% endif %}
    <div class="login-form">
        <form action="{% url 'login' %}" method="post">
            {{ form.as_p }}
            {% csrf_token %}
            <input type="hidden" name="next" value="{{ next }}" />
            <p><input type="submit" value="Log-in"></p>
        </form>
    </div>
</div>
{% endblock %}
```

این یک الگوی استاندارد برای ورود به سایت Django است.

الگوی ثبت نام / logged_out.html را ویرایش کنید و کد زیر را به آن اضافه کنید:

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

{% block title %}Logged out{% endblock %}

{% block content %}
<h1>Logged out</h1>
<div class="module">
    <p>You have been successfully logged out.
        You can <a href="{% url "login" %}">log-in again</a>.</p>
</div>
{% endblock %}
```

این الگوی است که پس از ورود به سیستم ، برای کاربر نمایش داده می شود.

سرور توسعه را با دستور زیر اجرا کنید:

```
python manage.py runserver
```

/http://127.0.0.1:8000/accounts/login را در مرورگر خود باز کنید. باید صفحه ورود را به این شکل مشاهده کنید:

Log-in

Please, use the following form to log-in:

Username:

Password:

LOGIN

ایجاد نماهای کلاس محور

ما می خواهیم برای ایجاد ، ویرایش و حذف دوره ها ، نماهای ایجاد کنیم. ما برای این کار از نماهای کلاس محور استفاده خواهیم کرد. پرونده views.py برنامه دوره را ویرایش کنید و کد زیر را به آن اضافه کنید:

```
from django.views.generic.list import ListView
from .models import Course

class ManageCourseListView(ListView):
    model = Course
    template_name = 'courses/manage/course/list.html'

    def get_queryset(self):
        qs = super(ManageCourseListView, self).get_queryset()
        return qs.filter(owner=self.request.user)
```

این نمای ManageCourseListView است. این از ListView عمومی Django به ارث می برد. ما برای دریافت تنها دوره های ایجاد شده توسط کاربر فعلی ، متده `get_queryset()` را رد می کنیم. برای جلوگیری از ویرایش ، به روزرسانی یا حذف دوره هایی که کاربران ایجاد نکرده اند ، باید به روش `get_queryset()` در ایجاد ، به روزرسانی و حذف نظرات نیز نادیده بگیریم. هنگامی که شما نیاز به ایجاد یک رفتار خاص برای چندین نمایش مبتنی بر کلاس دارید ، توصیه می شود از میکسین استفاده کنید.

استفاده از مخلوط ها برای نمایش کلاس مبتنی بر

میکسین نوع خاصی از ارث های متعدد برای یک کلاس است. شما می توانید از آنها برای ارائه قابلیت های گسته راچ استفاده کنید ، که به دیگر ترکیبات اضافه شده ، به شما امکان می دهد رفتار یک کلاس را تعریف کنید. دو حالت اصلی برای استفاده از ترکیب وجود دارد:

شما می خواهید چندین کلاس اختیاری را برای یک کلاس ارائه دهید شما می خواهید از چندین ویژگی خاص در چندین کلاس استفاده کنید. Django شامل چندین ترکیب است که عملکرد دیگری را برای نمایش های کلاس محور شما فراهم می کند. می توانید اطلاعات بیشتری در مورد mixins در <https://docs.djangoproject.com/en/2.0/topics/class-based-mixins/> کسب کنید.

ما قصد داریم یک کلاس mixin ایجاد کنیم که یک رفتار مشترک را شامل می شود و از آن برای دیدگاه های دوره استفاده می کند. پرونده views.py برنامه دوره را ویرایش کنید و آن را به شرح زیر اصلاح کنید:

```
from django.urls import reverse_lazy
from django.views.generic.list import ListView
from django.views.generic.edit import CreateView, UpdateView, \
    DeleteView
from .models import Course

class OwnerMixin(object):
    def get_queryset(self):
        qs = super(OwnerMixin, self).get_queryset()
        return qs.filter(owner=self.request.user)

class OwnerEditMixin(object):
    def form_valid(self, form):
        form.instance.owner = self.request.user
```

```

        return super(OwnerEditMixin, self).form_valid(form)

class OwnerCourseMixin(OwnerMixin):
    model = Course

class OwnerCourseEditMixin(OwnerCourseMixin, OwnerEditMixin):
    fields = ['subject', 'title', 'slug', 'overview']
    success_url = reverse_lazy('manage_course_list')
    template_name = 'courses/manage/course/form.html'

class ManageCourseListView(OwnerCourseMixin, ListView):
    template_name = 'courses/manage/course/list.html'

class CourseCreateView(OwnerCourseEditMixin, CreateView):
    pass

class CourseUpdateView(OwnerCourseEditMixin, UpdateView):
    pass

class CourseDeleteView(OwnerCourseMixin, DeleteView):
    template_name = 'courses/manage/course/delete.html'
    success_url = reverse_lazy('manage_course_list')

```

در این کد میکسین های OwnerEditMixin و OwnerMixin را ایجاد می کنیم. ما از این ترکیب ها به همراه نمایش های CreateView، ListView، UpdateView و DeleteView تهیه شده توسط Django استفاده خواهیم کرد. روشن زیر را پیاده سازی می کند:

(): این روش توسط نماها برای بدست آوردن پایه QuerySet استفاده می شود. ترکیب ما با استفاده از این روش برای فیلتر کردن اشیاء با استفاده از ویژگی مالک ، برای بازیابی اشیاء متعلق به کاربر فعلی (درخواست.وزر) از این روش استفاده می کند.

روشن زیر را پیاده سازی می کند:

(): این روش توسط نماهای استفاده از Django's ModelFormMixin mixin شود که از form_valid() استفاده می کند ، یعنی نمایی با فرم ها یا فرم های مدل مانند CreateView و UpdateView. وقتی فرم ارائه شده معتبر است ، فرم valid() اجرا می شود. رفتار پیش فرض برای این روش صرفه جویی در نمونه (برای فرم های مدل) و هدایت کاربر به موفقیت_ورل

است. ما این روش را رد می کنیم تا به طور خودکار کاربر فعلی را در ویژگی صاحب شیء ذخیره شده تنظیم کنیم. با این کار، در هنگام ذخیره، مالک را برای یک شی به صورت خودکار تنظیم می کنیم.

کلاس OwnerMixin ما می تواند برای نماهایی که با هر مدلی که دارای ویژگی خصوصیات باشد تعامل داشته باشد.

ما همچنین یک کلاس OwnerCourseMixin را تعریف می کنیم که از مالکیت OwnerMixin به ارث می رود و ویژگی های زیر را برای نماهای کودک ارائه می دهیم:

مدل: مدل مورد استفاده برای QuerySets. مورد استفاده توسط تمام نماها

ما یک ترکیبی از OwnerCourseEditMixin با ویژگی های زیر تعریف می کنیم:

فیلدها: زمینه های مدل برای ساختن فرم مدل نمایش CreateView و UpdateView.

ucces_url: پس از ارسال فرم با موفقیت، توسط CreateView و UpdateView برای تغییر مسیر کاربر استفاده می شود.
ما از یک URL با نام management_course_list استفاده می کنیم که بعداً می خواهیم ایجاد کنیم.

در آخر، ما دیدگاههای زیر را ایجاد می کنیم که زیر کلاس OwnerCourseMixin است:

ManageCourseListView: لیست دوره های ایجاد شده توسط کاربر را لیست می کند. این از OwnerCourseMixin و ListView به ارث می رسد.

CourseCreateView: از یک فرم مدل برای ایجاد یک شی جدید Course استفاده می کند.

CreatView: از فیلدهای تعریف شده در OwnerCourseEditMixin برای ساختن یک فرم مدل و همچنین زیر کلاس های استفاده می کند.

CourseUpdateView و OwnerCourseEditMixin: به ویرایش یک شیء موجود در Course اجازه می دهد. این از CourseUpdateView و UpdateView به ارث می رسد.

CourseDeleteView: ورا ث از OwnerCourseMixin را برای تغییر مسیر کاربر DeleteView. sukses_url و عمومی پس از حذف شیء تعریف می کند.

کار با گروه ها و مجوزها

ما برای مدیریت دوره ها دیدگاه های اساسی ایجاد کرده ایم. در حال حاضر ، هر کاربر می تواند به این نماها دسترسی پیدا کند. ما می خواهیم این دیدگاه ها را محدود کنیم تا فقط مربیان اجازه ایجاد دوره ها و مدیریت دوره ها را داشته باشند.

چارچوب احراز هویت Django شامل یک سیستم اجازه است که به شما امکان می دهد مجوزها را به کاربران و گروه ها اختصاص دهید. ما می خواهیم گروهی را برای کاربران مربی ایجاد کنیم و مجوزهایی را برای ایجاد ، به روزرسانی و حذف دوره ها اختصاص دهیم.

سرور توسعه را با استفاده از دستور اجرا کنید و <http://127.0.0.1:8000/admin/auth/group/add> را در مرورگر خود باز کنید تا یک هدف جدید گروه ایجاد شود. نام مربیان را اضافه کنید و تمام مجوزهای مربوط به دوره را به جز موارد مدل Subject ، به شرح زیر انتخاب کنید:

Add group

Name:

Permissions:

Available permissions (0)	Chosen permissions (0)
<input type="checkbox"/> Filter	
admin log entry Can add log entry	
admin log entry Can change log entry	
admin log entry Can delete log entry	
auth group Can add group	
auth group Can change group	
auth group Can delete group	
auth permission Can add permission	
auth permission Can change permission	
auth permission Can delete permission	
auth user Can add user	
auth user Can change user	

Hold down "Control", or "Command" on a Mac, to select more than one.

همانطور که مشاهده می کنید ، برای هر مدل سه مجوز مختلف وجود دارد: می توانید اضافه کنید ، تغییر دهید و حذف کنید. پس از انتخاب مجوز برای این گروه ، بر روی دکمه SAVE کلیک کنید.

Django به طور خودکار مجوزهایی را برای مدل ها ایجاد می کند ، اما همچنین می توانید مجوزهای سفارشی ایجاد کنید. می توانید اطلاعات بیشتر در مورد اضافه کردن مجوزهای سفارشی را در <https://docs.djangoproject.com/fa/2.0/topics/auth/customizing/#custom-permission> مطالعه کنید.

/ را باز کنید و یک کاربر جدید ایجاد کنید. کاربر را ویرایش کنید و آن را در گروه مربیان اضافه کنید ، به شرح زیر:

Groups:



کاربران مجوزهای گروههایی را که به آنها تعلق دارند به ارث می بردند ، اما می توانند با استفاده از سایت مدیریت ، مجوزهای فردی را نیز به یک کاربر اضافه کنید. کاربرانی که دارای `True` روی `is_superuser` هستند ، همه مجوزها را به صورت خودکار دارند.

محدود کردن دسترسی به نماهای مبتنی بر کلاس

ما می خواهیم دسترسی به نماها را محدود کنیم تا فقط کاربران دارای مجوزهای مناسب بتوانند اشیاء `Course` را اضافه یا تغییر دهند. ما می خواهیم از دو ترکیب زیر که توسط `django.contrib.auth` ارائه شده است استفاده کنیم تا دسترسی به بازدیدها محدود شود.

عملکرد دکوراتور login_required را نشان می دهد.

به کاربرانی که مجوز خاصی دارند اجازه دسترسی به نمای را می دهد. به یاد داشته باشید که بطور خودکار تمام مجوزها را دارد. superusers

پرونده views.py برنامه دوره را ویرایش کنید و واردات زیر را اضافه کنید:

```
from django.contrib.auth.mixins import LoginRequiredMixin, \
    PermissionRequiredMixin
```

را به ارث برسانید LoginRequiredMixin مانند این:

```
class OwnerCourseMixin(OwnerMixin, LoginRequiredMixin):
    model = Course
    fields = ['subject', 'title', 'slug', 'overview']
    success_url = reverse_lazy('manage_course_list')
```

سپس ، یک خصوصیت lev_required را به ایجاد ، بروزرسانی و حذف نمایش ها به شرح زیر اضافه کنید:

```
class CourseCreateView(PermissionRequiredMixin,
                      OwnerCourseEditMixin,
                      CreateView):
    permission_required = 'courses.add_course'

class CourseUpdateView(PermissionRequiredMixin,
                      OwnerCourseEditMixin,
                      UpdateView):
    permission_required = 'courses.change_course'

class CourseDeleteView(PermissionRequiredMixin,
                      OwnerCourseMixin,
                      DeleteView):
    template_name = 'courses/manage/course/delete.html'
    success_url = reverse_lazy('manage_course_list')
    permission_required = 'courses.delete_course'
```

بررسی می کند که کاربر می تواند به این نمایه دسترسی داشته باشد ، مجوز مشخص شده در خصوصیت `lev_required` را دارد. بازدیدهای ما اکنون فقط در دسترس کاربرانی است که دارای مجوزهای مناسب هستند.

بیایید URL را برای این نمایش ها ایجاد کنیم. یک پرونده جدید در داخل دوره ها ایجاد کنید.

دایرکتوری برنامه و نام آن `urls.py`. کد زیر را به آن اضافه کنید:

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

urlpatterns = [
    path('mine/',
        views.ManageCourseListView.as_view(),
        name='manage_course_list'),
    path('create/',
        views.CourseCreateView.as_view(),
        name='course_create'),
    path('<pk>/edit/',
        views.CourseUpdateView.as_view(),
        name='course_edit'),
    path('<pk>/delete/',
        views.CourseDeleteView.as_view(),
        name='course_delete'),
]
```

اینها الگوهای URL لیست ، ایجاد ، ویرایش و حذف نمرات دوره هستند. فایل اصلی urls.py پروژه آموزشی را ویرایش کنید و الگوهای URL برنامه دوره را زیر به شرح زیر وارد کنید:

```
from django.urls import path, include

urlpatterns = [
    path('accounts/login/', auth_views.LoginView.as_view(), name='login'),
    path('accounts/logout/', auth_views.LogoutView.as_view(), name='logout'),
    path('admin/', admin.site.urls),
    path('course/', include('courses.urls')),
]
```

ما باید الگوهای این نماها را ایجاد کنیم. دایرکتوری ها و پرونده های زیر را در قالب / فهرست راهنمای برنامه دوره ایجاد کنید:

```
courses/
    manage/
        course/
            list.html
            form.html
            delete.html
```

الگوی دوره ها / management / course / list.html را ویرایش کنید و کد زیر را به آن اضافه کنید:

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

{% block title %}My courses{% endblock %}

{% block content %}
<h1>My courses</h1>

<div class="module">
    {% for course in object_list %}
        <div class="course-info">
            <h3>{{ course.title }}</h3>
            <p>
                <a href="{% url "course_edit" course.id %}">Edit</a>
                <a href="{% url "course_delete" course.id %}">Delete</a>
            </p>
        </div>
    {% empty %}

```



```
        <p>You haven't created any courses yet.</p>
    {% endfor %}
    <p>
        <a href="{% url "course_create" %}" class="button">Create new
course</a>
    </p>
</div>
{% endblock %}
```

این الگوی برای نمایش ManageCourseListView است. در این الگو دوره های ایجاد شده توسط کاربر فعلی را لیست می کنیم. ما پیوندهایی را برای ویرایش یا حذف هر دوره و پیوندی برای ایجاد دوره های جدید قرار می دهیم.

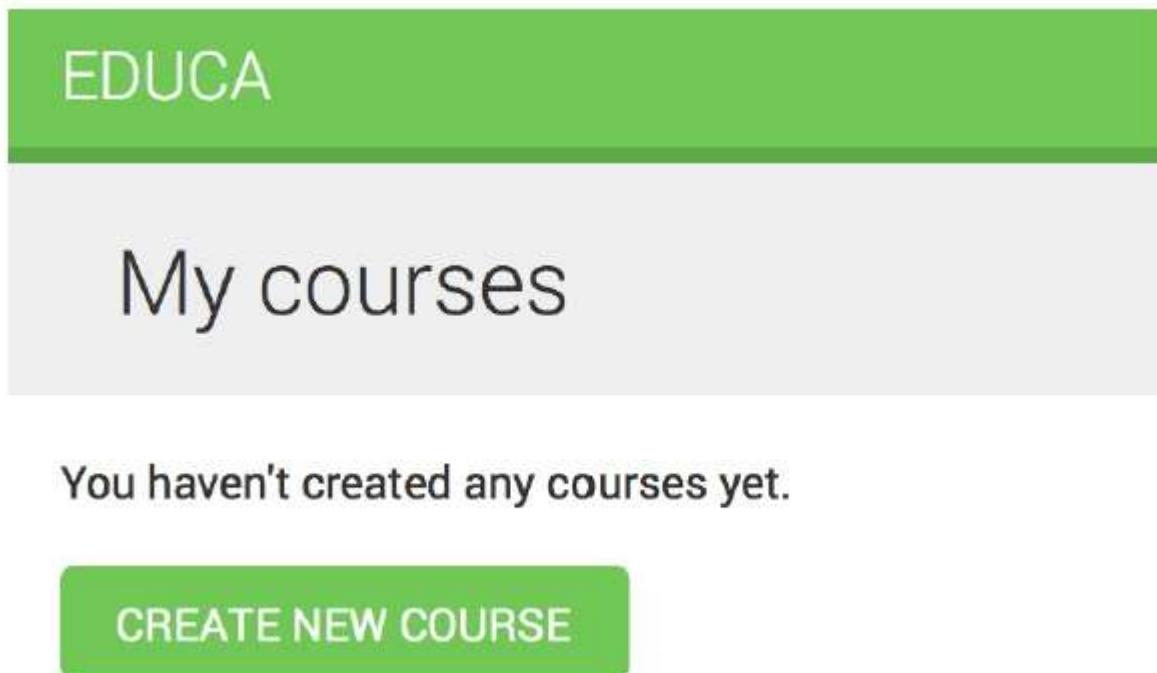
سرور توسعه را با استفاده از دستور `python management.py` اجرا کنید

سرور باز کن

`/http://127.0.0.1:8000/accounts/login/?next=/course/mine`

در مرورگر خود وارد شوید و با کاربری که به گروه مربیان تعلق دارد وارد شوید.

پس از ورود به سیستم ، به آدرس `http://127.0.0.1:8000/course/mine/` URL هدایت می شوید و باید موارد زیر را ببینید



در این صفحه کلیه دوره های ایجاد شده توسط کاربر فعلی نمایش داده می شود.
بیایید الگوی را ایجاد کنیم که فرم مربوط به نمایش و نمایش دوره های دوره را نشان می دهد. الگوی دوره ها /
بیایید الگوی را ایجاد کنیم که فرم مربوط به نمایش و نمایش دوره های دوره را نشان می دهد. الگوی دوره ها /
را ویرایش کنید و کد زیر را بنویسید:

```

{% extends "base.html" %}

{% block title %}
    {% if object %}
        Edit course "{{ object.title }}"
    {% else %}
        Create a new course
    {% endif %}
{% endblock %}

{% block content %}
<h1>
    {% if object %}
        Edit course "{{ object.title }}"
    {% else %}
        Create a new course
    {% endif %}
</h1>
<div class="module">
    <h2>Course info</h2>
    <form action=". " method="post">
        {{ form.as_p }}
        {% csrf_token %}
        <p><input type="submit" value="Save course"></p>
    </form>
</div>
{% endblock %}

```

قالب فرم برای هر دو نمایش CourseUpdateView و CourseCreateView استفاده می شود. در این الگو، بررسی می کنیم متغیر شیء در متن باشد یا نه. اگر شیء در متن وجود داشته باشد، می دانیم که یک دوره موجود را به روز می کنیم و از آن در عنوان صفحه استفاده می کنیم. در غیر این صورت، ما در حال ایجاد یک موضوع جدید Course هستیم.

در مرورگر خود CREATE NEW COURSE را باز کرده و روی دکمه /http://127.0.0.1:8000/course/mine کلیک کنید. صفحه زیر را مشاهده خواهید کرد:

Create a new course

Course info

Subject:

Title:

Slug:

Overview:

فرم را پر کرده و بر روی دکمه SAVE COURSE کلیک کنید. دوره ذخیره می شود و شما به صفحه لیست دوره ها هدایت می شوید. باید به شرح زیر باشد:

My courses

Django course

[Edit](#) [Delete](#)[CREATE NEW COURSE](#)

سپس بر روی پیوند ویرایش برای دوره ای که اخیراً ایجاد کرده اید کلیک کنید. دوباره فرم را مشاهده خواهید کرد ، اما این بار به جای ایجاد یک مورد ، در حال ویرایش یک شیء Course موجود هستید.

در آخر ،courses/manage/course/delete.html اضافه کنید:

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

{% block title %}Delete course{% endblock %}

{% block content %}
<h1>Delete course "{{ object.title }}"</h1>

<div class="module">
  <form action="" method="post">
    {% csrf_token %}
    <p>Are you sure you want to delete "{{ object }}"?</p>
    <input type="submit" class="button" value="Confirm">
  </form>
</div>
{% endblock %}
```

این الگوی برای نمایش CourseDeleteView است. این نمای از DeleteView ارائه شده توسط Django ارث می برد ،
که انتظار دارد تایید کاربر برای حذف یک شیء باشد.

مروگر خود را باز کرده و روی پیوند حذف دوره خود کلیک کنید. باید صفحه تأیید زیر را مشاهده کنید:

EDUCA

Delete course "Django course"

Are you sure you want to delete "Django course"?

CONFIRM

روی دکمه CONFIRM کلیک کنید. دوره حذف می شود و شما دوباره به صفحه لیست دوره ها هدایت می شوید.

مربیان هم اکنون می توانند دوره ها را ایجاد ، ویرایش و حذف کنند. در مرحله بعد ، ما باید CMS را برای آنها فراهم کنیم تا مازول ها و مطالبات را به دوره ها اضافه کنیم.

ما با مدیریت مازول های دوره شروع خواهیم کرد.

مدیریت مازول ها و محتوا

ما قصد داریم سیستمی را برای مدیریت مازول های دوره و محتویات آنها بسازیم. ما نیاز به ایجاد اشکال داریم که می تواند برای مدیریت چندین مازول در هر دوره و انواع مختلف محتوا برای هر مازول استفاده شود. هر دو مازول و محتوا باید از یک نظم خاص پیروی کنند و باید بتوانیم آنها را با استفاده از CMS ترتیب بندی مجدد کنیم.

استفاده از قالب ها برای مازول های دوره

جنگو با یک لایه انتزاع همراه است تا با چندین فرم در همان صفحه کار کند. این گروه از فرم ها به عنوان قالب شناخته می شوند.

قالب ها چندین نمونه از یک فرم خاص یا ModelForm را مدیریت می کنند. کلیه فرم ها به صورت یکجا ارسال می شوند و فرم فرم از تعداد اولیه فرم ها برای نمایش مراقبت می کند ، و حداقل تعداد فرم های ارسال شده را محدود می کند و اعتبار آن را تأیید می کند.

فرم ها شامل روشی is_valid() برای اعتبارسنجی همه فرم ها به طور همزمان هستند.

همچنین می توانید داده های اولیه را برای فرم ها تهیه کنید و چند شکل خالی اضافی را برای نمایش مشخص کنید.
می توانید اطلاعات بیشتر در مورد قالب ها را در <https://docs.djangoproject.com/fa/2> کسب کنید.

0 / موضوعات / فرمها / قالبها / و درباره قالب های مدل در <https://docs.djangoproject.com/en/2.0/topics/forms/modelformsets/#model-formsets>

از آنجاکه یک دوره به تعداد متغیر مأذول ها تقسیم می شود ، استفاده از قالب ها برای مدیریت آنها منطقی است. یک پرونده forms.py را در دایرکتوری برنامه دوره ها ایجاد کنید و کد زیر را به آن اضافه کنید:

```
from django import forms
from django.forms.models import inlineformset_factory
from .models import Course, Module

ModuleFormSet = inlineformset_factory(Course,
                                      Module,
                                      fields=['title',
                                               'description'],
                                      extra=2,
                                      can_delete=True)
```

این فرم قالب ModuleFormSet است. ما آن را با استفاده از تابع inlineformset_factory() ارائه شده توسط Django می سازیم. قالبهای درون خطی انتزاعی کوچک در بالای قالبها هستند که کار با اشیاء مرتبط را ساده می کنند. این عملکرد به ما امکان می دهد یک قالب مدل بسازیم

به صورت پویا برای اشیاء مأذول مربوط به یک موضوع Course.

ما از پارامترهای زیر برای ساخت قالب استفاده می کنیم:

فیلدهایی که در هر فرم از فرم قرار خواهند گرفت.

اضافی: به ما امکان می دهد تعداد فرم های اضافی خالی را برای نمایش در قالب تنظیم کنیم.

:اگر این گزینه را روی True تنظیم کنید ، یک فرم Boolean Django برای هر فرم که به عنوان ورودی جعبه چک ارائه می شود ، شامل می شود.

این امکان را به شما می دهد تا اشیایی را که می خواهید حذف کنید علامت گذاری کنید.

پرونده views.py برنامه دوره را ویرایش کنید و کد زیر را به آن اضافه کنید:

```
from django.shortcuts import redirect, get_object_or_404
from django.views.generic.base import TemplateResponseMixin, View
from .forms import ModuleFormSet

class CourseModuleUpdateView(TemplateResponseMixin, View):
    template_name = 'courses/manage/module/formset.html'
    course = None

    def get_formset(self, data=None):
        return ModuleFormSet(instance=self.course,
                             data=data)

    def dispatch(self, request, pk):
        self.course = get_object_or_404(Course,
                                         id=pk,
                                         owner=request.user)
        return super(CourseModuleUpdateView,
                    self).dispatch(request, pk)

    def get(self, request, *args, **kwargs):
```

```

        formset = self.get_formset()
        return self.render_to_response({'course': self.course,
                                       'formset': formset})

    def post(self, request, *args, **kwargs):
        formset = self.get_formset(data=request.POST)
        if formset.is_valid():
            formset.save()
            return redirect('manage_course_list')
        return self.render_to_response({'course': self.course,
                                       'formset': formset})

```

نمای CourseModuleUpdateView برای افزودن ، به روزرسانی و حذف مازول ها برای یک دوره خاص ، فرم را کنترل می کند. این نمای از ترکیبات و نظرات زیر به ارت می رسد:

: این ترکیب وظیفه ارائه الگوهای قالب و برگرداندن پاسخ HTTP را بر عهده می گیرد. به یک ویژگی TemplateResponseMixin نیاز دارد که الگوی ارائه شده را نشان می دهد و روش render_to_response() را برای گذاردن یک متن و ارائه الگو فراهم می کند.

نمای کلاس مبتنی بر کلاس که توسط Django ارائه شده است.

در این دیدگاه ، روشهای زیر را پیاده سازی می کنیم:

(): ما از این روش برای جلوگیری از تکرار کد برای ساختن قالب استفاده می کنیم. ما یک شیء ModuleFormSet برای شیء Course داده با داده های اختیاری ایجاد می کنیم.

(): این روش توسط کلاس View ارائه می شود. این درخواست HTTP و پارامترهای آن را انجام می دهد و تلاش می کند تا از یک روش کوچک که با روش HTTP استفاده شده مطابقت داشته باشد: یک درخواست GET به روش get() و یک درخواست POST برای ارسال () به آنها واگذار شده است. در این روش ما از تابع میانبر get_object_or_404() استفاده می کنیم تا این روش Course برای پارامتر id داده شده متعلق به کاربر فعلی استفاده کنیم. ما این کد را در روش ارسال () اعزام می کنیم زیرا ما نیاز به بازیابی دوره برای هر دو درخواست GET و POST داریم.

ما آن را در ویژگی دوره دیدگاه ذخیره می کنیم تا در دسترس سایر روشهای قرار گیرد.

() برای درخواست GET اعدام شد. ما با استفاده از متده `render_to_response()` ارائه شده توسط `TemplateResponseMixin` به `CourseModuleFormSet` یک قالب `ModuleFormSet` خالی ایجاد می کنیم و آن را به همراه شیء فعلی `Course` به الگوی ارائه می دهیم.

() برای درخواست های ارسال شده اعدام شد. در این روش اقدامات زیر را انجام می دهیم:

1. ما با استفاده از داده های ارسالی یک نمونه `ModuleFormSet` ایجاد می کنیم.

2. ما برای تأیید اعتبار کلیه اشکال آن ، روش `is_valid()` فرم را اجرا می کنیم.

۳- اگر فرمت معتبر است ، ما با فراخوانی روش `save()` ذخیره می کنیم. در این مرحله ، هرگونه تغییر ایجاد شده از جمله اضافه کردن ، به روزرسانی یا مارک های مارک برای حذف ، روی بانک اطلاعاتی اعمال می شود. سپس ، ما کاربران را به `URL_main_course_list` هدایت می کنیم. اگر فرمت معتبر نیست ، ما

در عوض ، قالب را برای نمایش هرگونه خطأ ارائه دهید.

پرونده `urls.py` برنامه دوره ها را ویرایش کنید و الگوی URL زیر را به آن اضافه کنید:

```
path('<pk>/module/',
     views.CourseModuleUpdateView.as_view(),
     name='course_module_update'),
```

یک دایرکتوری جدید در داخل دوره ها ایجاد کنید `courses/manage/template` را ایجاد کنید و مأذول آن را نامگذاری کنید. یک الگوی `courses/manage/module/formset.html` ایجاد کنید و کد زیر را به آن اضافه کنید:

```

{% extends "base.html" %}

{% block title %}
    Edit "{{ course.title }}"
{% endblock %}

{% block content %}
    <h1>Edit "{{ course.title }}"</h1>
    <div class="module">
        <h2>Course modules</h2>
        <form action="" method="post">
            {{ formset }}
            {{ formset.management_form }}
            {% csrf_token %}
            <input type="submit" class="button" value="Save modules">
        </form>
    </div>
{% endblock %}

```

در این الگویک عنصر `<form>` ایجاد می کنیم ، که در آن قالب قرار می دهیم. ما همچنین فرم مدیریتی برای قالب را با متغیر `{{ formatet.management_form }}` include کنیم. فرم مدیریت شامل فیلدهای پنهان برای کنترل تعداد اولیه ، کل ، حداقل و حداکثر تعداد فرم ها می باشد. می توانید ببینید که ایجاد قالب بسیار ساده است.

الگوی دوره ها / `management / course / list.html` را ویرایش کنید و پیوند زیر را برای آدرس URL در زیر دوره پیوند ویرایش و حذف کنید:

```

<a href="{% url "course_edit" course.id %}">Edit</a>
<a href="{% url "course_delete" course.id %}">Delete</a>
<a href="{% url "course_module_update" course.id %}">Edit
modules</a>

```

ما پیوند را برای ویرایش مازول های دوره درج کرده ایم. `/http://127.0.0.1:8000/course/mine/` را در مرورگر خود باز کنید. یک دوره ایجاد کنید و روی پیوند ویرایش مازول ها برای آن کلیک کنید. شما باید یک فرم را به شرح زیر مشاهده کنید:

Edit "Django course"

Course modules

Title:

Description:

Delete:

Title:

Description:

Delete:

SAVE MODULES

قالب شامل یک فرم برای هر شیء مازول موجود در این دوره است. پس از اینها ، دو فرم اضافی خالی نمایش داده می شود زیرا ما 2 برای ModuleFormSet تنظیم می کنیم. هنگامی که قالب را ذخیره می کنید ، Django شامل دو قسمت اضافی دیگر برای اضافه کردن مازول های جدید می شود.

[افزودن محتوا به مازول های دوره](#)

اکنون به روشی برای اضافه کردن مطالب به مژول های دوره نیاز داریم. ما چهار نوع مختلف از مطالب داریم: متن ، فیلم ، تصویر و پرونده. برای ایجاد محتوا می توان چهار نمای مختلف را در نظر گرفت ، یکی برای هر مدل. با این حال ما می خواهیم رویکردی کلی تر به وجود آوریم و دیدگاهی ایجاد کنیم که اشیاء هر مدل محتوا را به روز می کند.

پرونده views.py برنامه دوره را ویرایش کنید و کد زیر را به آن اضافه کنید:

```
from django.forms.models import modelform_factory
from django.apps import apps
from .models import Module, Content

class ContentCreateUpdateView(TemplateResponseMixin, View):
    module = None
    model = None
    obj = None
    template_name = 'courses/manage/content/form.html'

    def get_model(self, model_name):
        if model_name in ['text', 'video', 'image', 'file']:
            return apps.get_model(app_label='courses',
                                  model_name=model_name)
        return None

    def get_form(self, model, *args, **kwargs):
        Form = modelform_factory(model, exclude=['owner',
                                                'order',
                                                'created',
                                                'updated'])
        return Form(*args, **kwargs)

    def dispatch(self, request, module_id, model_name, id=None):
        self.module = get_object_or_404(Module,
```

```
        id=module_id,
        course_owner=request.user)
self.model = self.get_model(model_name)
if id:
    self.obj = get_object_or_404(self.model,
                                 id=id,
                                 owner=request.user)
return super(ContentCreateUpdateView,
            self).dispatch(request, module_id, model_name, id)
```

این قسمت اول ContentCreateUpdateView است. این امکان را به ما می دهد تا محتویات مدل های مختلف را ایجاد و به روز کنیم. این نمای روش های زیر را تعریف می کند:

get_model(): در اینجا ، بررسی می کنیم که نام مدل داده شده یکی از چهار مدل محتوا است: متن ، ویدئو ، تصویر یا پرونده. سپس ، ما از مازول برنامه Django استفاده می کنیم تا کلاس واقعی برای نام مدل داده شده را بدست آوریم. اگر نام مدل داده شده یکی از موارد معتبر نباشد ، ما None را برمی گردیم.

get_form(): ما با استفاده از تابع modelform_fective() چارچوب فرم ، فرم پویایی می سازیم. از آنجایی که می خواهیم یک فرم برای مدل های Text ، Video و File بسازیم ، از پارامتر exclude برای مشخص کردن قسمت های مشترک استفاده می کنیم تا از فرم خارج شویم و اجازه می دهیم که تمام خصوصیات دیگر بطور خودکار درج شوند. با این کار ، لازم نیست بدانیم بسته به مدل ، کدام قسمتها را شامل شود.

اعزام(): پارامترهای URL زیر را دریافت می کند و مازول ، مدل و شیء محتوا را به عنوان ویژگی های کلاس ذخیره می کند: module_id: شناسه مازولی که محتوا با آن در ارتباط است.

model_name: نام مدل محتوا برای ایجاد / بروزرسانی.

id: شناسه شیئی که به روز می شود. این هیچکدام برای ایجاد اشیاء جدید نیست.

روشهای دریافت و ارسال زیر را به ContentCreateUpdateView اضافه کنید:

```

def get(self, request, module_id, model_name, id=None):
    form = self.get_form(self.model, instance=self.obj)
    return self.render_to_response({'form': form,
                                   'object': self.obj})

def post(self, request, module_id, model_name, id=None):
    form = self.get_form(self.model,
                         instance=self.obj,
                         data=request.POST,
                         files=request.FILES)
    if form.is_valid():
        obj = form.save(commit=False)
        obj.owner = request.user
        obj.save()
        if not id:
            # new content
            Content.objects.create(module=self.module,
                                   item=obj)
    return redirect('module_content_list', self.module.id)

    return self.render_to_response({'form': form,
                                   'object': self.obj})

```

این روش ها به شرح زیر است:

get () : وقتی درخواست GET دریافت شد ، اجرا می شود. ما فرم نمونه را برای نمونه متن ، ویدئو ، تصویر یا پرونده که به روز می شود ساختیم. در غیر این صورت ، ما هیچ نمونه ای را برای ایجاد یک شی جدید نمی گذاریم ، زیرا اگر شناسه ای ارائه نشود ، self.obj نیست.

post () : هنگام دریافت درخواست POST اعدام می شود. ما modelform را با گذر از داده ها و پرونده های ارسال شده به آن می سازیم. سپس ، آن را تأیید می کنیم. اگر فرم معتبر باشد ، ما قبل از ذخیره کردن آن ، یک شی جدید ایجاد می کنیم و درخواست را به عنوان مالک آن اختصاص می دهیم

بانک اطلاعاتی ما پارامتر id را بررسی می کنیم. اگر هیچ شناسه ای ارائه نشود ، می دانیم کاربر به جای به روزرسانی موجود ، در حال ایجاد یک شی جدید است. اگر این یک شی جدید باشد ، ما یک شیء Content را برای ماثول داده شده ایجاد می کنیم و محتوای جدید را به آن مرتبط می کنیم.

پرونده urls.py برنامه دوره را ویرایش کنید و الگوهای URL زیر را به آن اضافه کنید:

```
path('module/<int:module_id>/content/<model_name>/create/',
      views.ContentCreateUpdateView.as_view(),
      name='module_content_create'),  
  
path('module/<int:module_id>/content/<model_name>/<id>/',
      views.ContentCreateUpdateView.as_view(),
      name='module_content_update'),
```

الگوهای جدید URL به شرح زیر است:

: برای ایجاد متن ، ویدئو ، تصویر یا پرونده های جدید و اضافه کدن آنها به یک ماژول. این شامل پارامترهای module_id و model_name است. مورد اول این امکان را می دهد که شیء محتوای جدید را به ماژول داده شده پیوند دهید. حالت دوم برای ساختن فرم ، مدل محتوا را مشخص می کند.

: برای به روزرسانی متن ، ویدئو ، تصویر یا پرونده موجود موجود. این شامل پارامترهای module_content_update و یک پارامتر id برای شناسایی محتوای به روز شده است.

یک دایرکتوری جدید در داخل دوره ها ایجاد کنید courses/manage/template را ایجاد کنید و نام آن را محتوا بگذارد. دوره های courses/manage/content/form.html را ایجاد کنید

و کد زیر را به آن اضافه کنید:

```

{% extends "base.html" %}

{% block title %}
    {% if object %}
        Edit content "{{ object.title }}"
    {% else %}
        Add a new content
    {% endif %}
{% endblock %}

{% block content %}
<h1>
    {% if object %}
        Edit content "{{ object.title }}"
    {% else %}
        Add a new content
    {% endif %}
</h1>
<div class="module">
    <h2>Course info</h2>
    <form action="" method="post" enctype="multipart/form-data">
        {{ form.as_p }}
        {% csrf_token %}
        <p><input type="submit" value="Save content"></p>
    </form>
</div>
{% endblock %}

```

این الگوی برای نمایش ContentCreateUpdateView است. در این الگو، بررسی می کنیم متغیر شیء در متن باشد یا نه. اگر یک شیء در متن وجود داشته باشد، ما یک شیء موجود را به روز می کنیم.

در غیر این صورت، ما در حال ایجاد یک شی جدید هستیم.

ما `form` را در عنصر `HTML` قرار می دهیم.
زیرا فرم شامل آپلود فایل برای مدل های محتوای فایل و تصویر است.

سرور توسعه را اجرا کنید، `http://127.0.0.1:8000/course/mine` را باز کنید،
برای یک دوره موجود بر روی ویرایش مازول ها کلیک کنید و یک مازول ایجاد کنید.

پوسته پایتون را با فرمان Python management.py باز کنید و شناسه جدیدترین ماثول ایجاد شده را به شرح زیر دریافت کنید:

```
>>> from courses.models import Module  
>>> Module.objects.latest('id').id  
6
```

سرور توسعه را اجرا کنید و http://127.0.0.1:8000/course/module/6/content/image/create را در مرورگر خود باز کنید و شناسه ماثول را با شخصی که قبلًا به دست آورده اید جایگزین کنید.

فرم ایجاد یک شی تصویر را به شرح زیر مشاهده خواهید کرد:

Add a new content

Course info

Title:

File:

no file selected

SAVE CONTENT

فرم را هنوز ارسال نکنید اگر سعی کنید این کار را انجام دهید ، شکست می خورد زیرا ما هنوز URL module_content_list را تعریف نکرده ایم. ما قصد داریم کمی آن را ایجاد کنیم.

برای حذف محتوا نیز نیاز به نمایش داریم. فایل views.py برنامه دوره را ویرایش کنید و کد زیر را اضافه کنید:

```
class ContentDeleteView(View):

    def post(self, request, id):
        content = get_object_or_404(Content,
                                    id=id,
                                    module_course_owner=request.user)
        module = content.module
        content.item.delete()
        content.delete()
        return redirect('module_content_list', module.id)
```

کلاس ContentDeleteView شیء Content را با شناسه مشخص بازیابی می کند. متن ، فیلم ، تصویر یا پرونده مربوط را حذف می کند. در آخر اینکه ، شیء Content را پاک می کند و کاربر را به آدرس URL module_content_list تغییر می دهد تا سایر مطالب مازول را لیست کند.

پرونده urls.py برنامه دوره را ویرایش کنید و الگوی URL زیر را به آن اضافه کنید:

```
path('content/<int:id>/delete/',
      views.ContentDeleteView.as_view(),
      name='module_content_delete'),
```

اکنون ، مربیان می توانند مطالب را به راحتی ایجاد ، بروزرسانی و حذف کنند.

[مدیریت مازول ها و محتويات](#)

ما برای ایجاد، ویرایش و حذف مأژول‌ها و محتویات دوره دیدهایی ایجاد کرده‌ایم. اکنون، ما به یک نمایش نیاز داریم تا همه مأژول‌ها را برای یک دوره نمایش دهیم و مطالب را برای یک مأژول خاص لیست کنیم.

پرونده views.py برنامه دوره را ویرایش کنید و کد زیر را به آن اضافه کنید:

```
class ModuleContentListView(TemplateResponseMixin, View):
    template_name = 'courses/manage/module/content_list.html'

    def get(self, request, module_id):
        module = get_object_or_404(Module,
                                   id=module_id,
                                   course__owner=request.user)

        return self.render_to_response({'module': module})
```

این ModuleContentListView است. این نمای با شناسه داده شده متعلق به کاربر فعلی می‌شود و با مأژول داده شده الگوی را ارائه می‌دهد.

پرونده urls.py برنامه دوره را ویرایش کنید و الگوی URL زیر را به آن اضافه کنید:

```
path('module/<int:module_id>',
      views.ModuleContentListView.as_view(),
      name='module_content_list'),
```

یک الگوی جدید در templates/courses/manage/module/content_list.html ایجاد کنید فهرست کنید و آن را content_list.html بنامید. کد زیر را به آن اضافه کنید:

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

{% block title %}
    Module {{ module.order|add:1 }}: {{ module.title }}
{% endblock %}

{% block content %}
    {% with course=module.course %}
        <h1>Course "{{ course.title }}"</h1>
        <div class="contents">
            <h3>Modules</h3>
            <ul id="modules">
                {% for m in course.modules.all %}
                    <li data-id="{{ m.id }}" {% if m == module %} class="selected" {% endif %}>
                        <a href="{% url "module_content_list" m.id %}">
                            <span>
                                Module <span class="order">{{ m.order|add:1 }}</span>
                            </span>
                            <br>
                            {{ m.title }}
                        </a>
                    </li>
                {% empty %}
                    <li>No modules yet.</li>
                {% endfor %}
            </ul>
        {% endwith %}
    {% endblock %}
```

```
<p><a href="{% url "course_module_update" course.id %}">  
    Edit modules</a></p>  
</div>  
<div class="module">  
    <h2>Module {{ module.order|add:1 }}: {{ module.title }}</h2>  
    <h3>Module contents:</h3>  
  
    <div id="module-contents">  
        {% for content in module.contents.all %}  
            <div data-id="{{ content.id }}">  
                {% with item=content.item %}  
                    <p>{{ item }}</p>  
                    <a href="#">Edit</a>  
                    <form action="{% url "module_content_delete" content.id %}"  
                        method="post">  
                        <input type="submit" value="Delete">  
                        {% csrf_token %}  
                    </form>  
                {% endwith %}  
            </div>  
        {% empty %}  
        <p>This module has no contents yet.</p>  
    
```

```
    {% endfor %}  
</div>  
    <h3>Add new content:</h3>  
    <ul class="content-types">  
        <li><a href="{% url "module_content_create" module.id "text" %}">  
            Text</a></li>  
        <li><a href="{% url "module_content_create" module.id "image" %}">  
            Image</a></li>  
        <li><a href="{% url "module_content_create" module.id "video" %}">  
            Video</a></li>  
        <li><a href="{% url "module_content_create" module.id "file" %}">  
            File</a></li>  
    </ul>  
</div>  
    {% endwith %}  
    {% endblock %}
```

این الگوی است که کلیه مارژول ها را برای یک دوره و محتویات مارژول انتخاب شده نمایش می دهد. ما در طول مارژول های دوره تکرار می کنیم تا آنها را در یک نوار کناری نمایش دهیم. ما برای دریافت متن ، ویدئو ، تصویر یا پرونده مربوط به محتوای مارژول تکرار می کنیم و به محتوای item دسترسی پیدا می کنیم.

ما همچنین پیوندهایی را برای ایجاد متن ، ویدئو ، تصویر یا محتوای جدید فایلها ایجاد می کنیم.

ما می خواهیم بدانیم که هر یک از اشیاء مورد کدام نوع است: متن ، فیلم ، تصویر یا پرونده. برای ساختن URL برای ایجاد URL به نام مدل احتیاج داریم. علاوه بر این ، ما می توانیم هر آیتم را بر اساس نوع محتوا در قالب مختلف نمایش دهیم. ما می توانیم با دسترسی به صفت meta_ شیء ، از یک کلاس متأ مدل بدست آوریم. با این وجود ، جنگو اجازه نمی دهد تا از متغیرها یا ویژگیهایی که با ایجاد نمرات در قالب ها شروع می شوند ، برای جلوگیری از بازیابی ویژگی های خصوصی یا تماس با روش های خصوصی استفاده کند. ما می توانیم با نوشتن یک فیلتر قالب سفارشی این مسئله را حل کنیم.

ساختار پرونده زیر را در فهرست برنامه کاربردی دوره ایجاد کنید:

```
templatetags/  
    __init__.py  
course.py
```

مارژول course.py را ویرایش کنید و کد زیر را به آن اضافه کنید:

```
from django import template  
  
register = template.Library()  
  
@register.filter  
def model_name(obj):  
    try:  
        return obj._meta.model_name  
    except AttributeError:  
        return None
```

این فیلتر قالب model_name است. می توانیم آن را در قالب ها به عنوان object | model_name اعمال کنیم تا نام مدل را برای یک شی دریافت کنیم.

% templates/courses/manage/module/content_list.html

کنید: % extends اضافه

```
{% load course %}
```

این برجسب های الگوی دوره باگیری می شود. سپس خطوط زیر را جایگزین کنید:

```
<p>{{ item }}</p>
<a href="#">Edit</a>
```

آنها را با موارد زیر جایگزین کنید:

```
<p>{{ item }} ({{ item|model_name }})</p>
<a href="{% url "module_content_update" module.id item|model_name item.id %}">Edit</a>
```

حال مدل مورد را در قالب نمایش می دهیم و از نام مدل برای ساخت پیوند برای ویرایش شی استفاده می کنیم. الگوی دوره ها / ویرایش کنید و پیوندی به آدرس management / course / list.html مانند این

اضافه کنید:

```
<a href="{% url "course_module_update" course.id %}">Edit modules</a>
{% if course.modules.count > 0 %}
  <a href="{% url "module_content_list" course.modules.first.id %}">
    Manage contents</a>
{% endif %}
```

پیوند جدید به کاربران این امکان را می دهد تا در صورت وجود به محتویات مازول اول دوره دسترسی پیدا کنند.

/ http://127.0.0.1:8000/course/mine را باز کنید و برای یک دوره که شامل حداقل یک مازول است، روی پیوند مدیریت محتوا کلیک کنید. صفحه ای مانند صفحه زیر را مشاهده خواهید کرد:

Course "Django course"

Modules

MODULE 1
Introduction to Django

[Edit modules](#)

Module 1: Introduction to Django

Module contents:

This module has no contents yet.

Add new content:

[Text](#)

[Image](#)

[Video](#)

[File](#)

هنگامی که بر روی یک مازول در نوار کناری سمت چپ کلیک می کنید ، محتوای آن در قسمت اصلی نمایش داده می شود. این الگو همچنین شامل پیوندهایی برای اضافه کردن متن ، ویدئو ، تصویر یا محتوای پرونده جدید برای نمایش مازول است. چند نوع محتوای مختلف را به مازول اضافه کنید و به نتیجه نگاهی بیندازید. مطالب بعد از مطالب مازول مانند مثال زیر ظاهر می شوند:

Course "Django course"

Modules

MODULE 1
Introduction to Django

MODULE 2
Configuring Django

[Edit modules](#)

Module 2: Configuring Django

Module contents:

Setting up Django (text)

[Edit](#)

[Delete](#)

Example settings.py (image)

[Edit](#)

[Delete](#)

Add new content:

[Text](#)

[Image](#)

[Video](#)

[File](#)

تنظیم مجدد مازول ها و مطالب

ما باید یک روش ساده برای تغییر ترتیب مازول های دوره و محتوای آنها ارائه دهیم. ما از یک ابزارک drag-n-drop جاوا اسکریپت استفاده خواهیم کرد تا به کاربران اجازه دهید با کشیدن آنها ، مازول های یک دوره را دوباره ترتیب دهند. هنگامی که کاربران کشیدن یک مازول را به پایان رساند ، یک درخواست ناهمزن (AJAX) برای ذخیره سفارش جدید مازول راه اندازی می کنیم.

Using mixins from djangobraces

یک ماثول شخص ثالث است که شامل مجموعه ای از ترکیب های عمومی برای Django است. این ترکیبات ویژگی های دیگری را برای نمای کلاس ارائه می دهد. لیستی از کلیه ترکیبات تهیه شده توسط django-braces را می توانید در اینجا مشاهده کنید

[./https://django-braces.readthedocs.io](https://django-braces.readthedocs.io)

ما از ترکیبات زیر بندهای django استفاده خواهیم کرد:

CsrfExemptMixin: برای جلوگیری از بررسی نشانه CSRF در درخواست های POST. ما برای انجام درخواستهای AJAX بدون نیاز به ایجاد csrf_token به این نیاز داریم.

JsonRequestResponseMixin: داده درخواست را به عنوان JSON تجزیه می کند و همچنین پاسخ را به عنوان JSON سریالی می کند و پاسخ HTTP را با نوع برنامه / json محتوای برمی گرداند.

django-braces را از طریق پیپ با استفاده از دستور زیر نصب کنید:

```
pip install django-braces
```

ما به نمایی نیاز داریم که سفارش جدید شناسه ماثول های رمزگذاری شده در JSON را دریافت کند. پرونده views.py برنامه دوره را ویرایش کنید و کد زیر را به آن اضافه کنید:

```
from braces.views import CsrfExemptMixin, JsonRequestResponseMixin

class ModuleOrderView(CsrfExemptMixin,
                     JsonRequestResponseMixin,
                     View):
    def post(self, request):
        for id, order in self.request_json.items():
            Module.objects.filter(id=id,
                                  course__owner=request.user).update(order=order)
        return self.render_json_response({'saved': 'OK'})
```

این نمای ModuleOrderView است.

ما می توانیم یک نظریه مشابه برای سفارش مطالب مژول بسازیم. کد زیر را به پرونده views.py اضافه کنید:

```
class ContentOrderView(CsrfExemptMixin,
                      JsonRequestResponseMixin,
                      View):
    def post(self, request):
        for id, order in self.request_json.items():
            Content.objects.filter(id=id,
                                   module_course__owner=request.user) \
                .update(order=order)
        return self.render_json_response({'saved': 'OK'})
```

اکنون پرونده urls.py برنامه دوره را ویرایش کنید و الگوهای URL زیر را به آن اضافه کنید:

```
path('module/order/',
      views.ModuleOrderView.as_view(),
      name='module_order'),  
  
path('content/order/',
      views.ContentOrderView.as_view(),
      name='content_order'),
```

سرانجام ، ما باید عملکرد drag-n-drop را در قالب پیاده سازی کنیم. ما برای این کار از کتابخانه jQuery UI استفاده خواهیم کرد. jQuery UI در بالایjQuery ساخته شده است و مجموعه ای از تعامل ، افکت ها و ابزارک رابط را در اختیار شما قرار می دهد. ما از عنصر قابل مرتب سازی آن استفاده خواهیم کرد. ابتدا باید jQuery UI را در الگوی پایه بارگذاری کنیم. پرونده base.html را که در قالب ها / دایرکتوری برنامه قرار دارد ، باز کنید و jQuery UI را در زیر اسکریپت اضافه کنید تا jQuery به شرح زیر بارگذاری شود:

```
<script  
src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js">  
</script>  
<script src="https://ajax.googleapis.com/ajax/libs/jqueryui/1.12.1/jquery-  
ui.min.js"></script>
```

ما بلا فاصله پس از چارچوب jQuery کتابخانه jQuery UI را بارگذاری می کنیم.

اکنون الگوی دوره ها / management / module / content_list.html را ویرایش کرده و کد زیر را در انتهای قالب اضافه کنید:

```
{% block domready %}  
  $('#modules').sortable({  
    stop: function(event, ui) {  
      modules_order = {};  
      $('#modules').children().each(function(){  
        // update the order field  
        $(this).find('.order').text($(this).index() + 1);  
        // associate the module's id with its order  
        modules_order[$(this).data('id')] = $(this).index();  
      });  
      $.ajax({  
        type: 'POST',  
        url: '{% url "module_order" %}',  
        contentType: 'application/json; charset=utf-8',  
        dataType: 'json',  
        data: JSON.stringify(modules_order)  
      });  
    }  
  });  
  
  $('#module-contents').sortable({  
    stop: function(event, ui) {  
      contents_order = {};  
      $('#module-contents').children().each(function(){  
        // associate the module's id with its order  
        contents_order[$(this).data('id')] = $(this).index();  
      });  
  
      $.ajax({  
        type: 'POST',  
        url: '{% url "content_order" %}',  
        contentType: 'application/json; charset=utf-8',  
        dataType: 'json',  
        data: JSON.stringify(contents_order),  
      });  
    }  
  });
```

```
});  
{% endblock %}
```

این کد JavaScript در بلوک `$(document).ready()` است و بنابراین در `base.html` تعریف کردیم گنجانده شده است. این تصمیم می کند که کد جاوا اسکریپت ما پس از بارگیری صفحه اجرا می شود. ما یک عنصر قابل مرتب سازی برای لیست مژول ها در نوار کناری و یک مورد دیگر برای لیست محتوای مژول تعریف می کنیم. هر دو به روش مشابه کار می کنند. در این کد کارهای زیر را انجام می دهیم:

1. ابتدا برای عنصر HTML مژول های قابل مرتب سازی تعریف می کنیم. به یاد داشته باشید که ما از `#modules` استفاده می کنیم، زیرا `jQuery` از نمادهای CSS برای انتخاب کنندگان استفاده می کند.
2. ما یک تابع برای رویداد توقف را مشخص می کنیم. این رویداد هر بار کاربر مرتب سازی مرتب سازی یک عنصر آغاز می شود.
3. ما یک فرهنگ لغت `modules_order` خالی ایجاد می کنیم. کلیدهای این فرهنگ لغت، شناسه مژول ها خواهد بود و مقادیر، ترتیب اختصاص یافته برای هر مژول خواهد بود.
4. ما بیش از عناصر `#module` کودکان تکرار می کنیم. ما ترتیب نمایش داده شده را برای هر مژول محاسبه می کنیم و ویژگی داده-شناسه آن، که حاوی شناسه مژول است را بدست می آوریم. ما `ID` را به عنوان کلید واژه نامه `modules_order` و فهرست جدید مژول به عنوان مقدار اضافه می کنیم.
- 5- درخواست POST AJAX را به `_URL` محتوا، از جمله داده های سریالی شده JSON `modules_order` در درخواست، راه اندازی می کنیم. `ModuleOrderView` مربوطه از بروزرسانی سفارش مژول ها مراقبت می کند.

عنصر قابل مرتب سازی برای سفارش مطالب کاملاً شبیه به این مورد است. برو

به مرورگر خود برگردید و صفحه را بازگیری مجدد کنید. اکنون ، شما می توانید هر دو مژول و محتوا را کلیک کنید و بکشید تا آنها را مانند مثال زیر ترتیب دهید:

The screenshot shows the EDUCA platform interface. At the top, there's a green header bar with the word "EDUCA". Below it, a large title says "Course 'Django course'". On the left, there's a sidebar titled "Modules" with two items: "MODULE 1 Introduction to Django" and "MODULE 2 Configuring Django". Below the sidebar, a green button says "Edit modules". The main content area shows "Module 2: Configuring Django" expanded. It has a "Contents:" section with two items: "Setting up Django (text)" and "Example settings.py (image)". Each item has "Edit" and "Delete" buttons below it. There's also a "Add new content:" button at the bottom of this section.

عالی! اکنون می توانید هر دو مژول دوره و محتوای مژول را مرتب کنید.

خلاصه

در این فصل یاد گرفتید که چگونه یک CMS همه کاره ایجاد کنید. شما از ارث مدل استفاده کرده اید و یک قسمت مدل دلخواه ایجاد کرده اید. شما همچنین با نماها و میکس های مبتنی بر کلاس کار کرده اید. شما قالبها و سیستمی را برای مدیریت انواع متنوعی از محتوا ایجاد کردید.

در فصل بعد یک سیستم ثبت نام دانشجویی ایجاد خواهد کرد.

شما همچنین انواع مختلفی از مطالب را ارائه می دهید ، و یاد می گیرید که چگونه با چارچوب حافظه نهان انفرادی Django کار کنید.

فصل یازدهم

ارائه و ذخیره محتوا

در فصل قبل شما از الگوهای ارثی و روابط عمومی برای ایجاد دوره های انعطاف پذیر ، مدل های محتوا استفاده کردید. شما همچنین با استفاده از نماهای مبتنی بر کلاس ، قالب ها و سفارش AJAX برای محتويات ، سیستم مدیریت دوره ساخته اید. در این فصل:

نمایش عمومی برای نمایش اطلاعات دوره ایجاد کنید

یک سیستم ثبت نام دانشجویی بسازید

ثبت نام دانشجو را در دوره ها مدیریت کنید

مطالب متنوع متن را ارائه دهید

محتوا حافظه پنهان با استفاده از چارچوب حافظه نهان

ما با ایجاد یک فهرست دوره ای برای دانشجویان برای مرور دوره های موجود و امکان ثبت نام در آنها شروع خواهیم کرد.

نمایش دوره ها

برای کاتالوگ دوره ما باید عملکردهای زیر را ایجاد کنیم:

لیست تمام دروس موجود ، که به صورت اختیاری توسط موضوع فیلتر شده است

نمای کلی دوره را نمایش دهید

فایل `views.py` برنامه دوره را ویرایش کنید و کد زیر را اضافه کنید:

```
from django.db.models import Count
from .models import Subject

class CourseListView(TemplateResponseMixin, View):
    model = Course
    template_name = 'courses/course/list.html'

    def get(self, request, subject=None):
        subjects = Subject.objects.annotate(
            total_courses=Count('courses'))
        courses = Course.objects.annotate(
            total_modules=Count('modules'))
        if subject:
            subject = get_object_or_404(Subject, slug=subject)
            courses = courses.filter(subject=subject)
        return self.render_to_response({'subjects': subjects,
                                        'subject': subject,
                                        'courses': courses})
```

این نمای CourseListView است. این از TemplateResponseMixin و View به ارث می رسد. در این نمای ، کارهای زیر را انجام می دهیم:

1. ماکلیه موضوعات ، از جمله تعداد کل دوره ها را برای هر یک از آنها بازیابی می کنیم. ما از حاشیه نویسی ORM استفاده می کنیم

1. ماکلیه موضوعات ، از جمله تعداد کل دوره ها را برای هر یک از آنها بازیابی می کنیم. ما از متدهای حاشیه نویسی () با عملکرد جمع (تعداد) استفاده می کنیم تا تعداد کل دوره ها را برای هر موضوع درج کنیم.

2. ماکلیه دوره های موجود ، از جمله تعداد کل مأذون های موجود در هر دوره را بازیابی می کنیم.

3. اگر یک پارامتر slug URL داده شده است ، ما موضوع مربوطه را بازیابی می کنیم و query را به دوره هایی که متعلق به موضوع مشخص هستند محدود می کنیم.

4- ما از روش render_to_response () ارائه شده توسط TemplateResponseMixin استفاده می کنیم تا اشیاء را به یک قالب تبدیل کرده و یک پاسخ HTTP را برگردانیم.

باید یک نمایش جزئیات برای نمایش یک مرور کلی در دوره ایجاد کنیم.

کد زیر را به پرونده views.py اضافه کنید:

```
from django.views.generic.detail import DetailView

class CourseDetailView(DetailView):
    model = Course
    template_name = 'courses/course/detail.html'
```

این دیدگاه از `DetailView` عمومی ارائه شده توسط Django ارث می برد.

ما ویژگی و مدل و قالب_نام را مشخص می کنیم. DetangoView Django

انتظار دارد که یک کلید اصلی (pk) یا یک پارامتر `slug` برای بازیابی یک شی واحد برای مدل داده شده باشد. سپس ، الگوی مشخص شده در `pattern_name` ، از جمله شیء موجود در متن را به عنوان یک شیء ارائه می دهد.

پرونده اصلی `urls.py` پروژه آموزشی را ویرایش کنید و الگوی URL زیر را به آن اضافه کنید:

```
from courses.views import CourseListView

urlpatterns = [
    # ...
    path('', CourseListView.as_view(), name='course_list'),
]
```

الگوی URL `course_list` URL را به پرونده اصلی `urls.py` پروژه اضافه می کنیم زیرا می خواهیم لیست دوره ها را در URL / نمایش دهیم و کلیه آدرس های اینترنتی دیگر برای برنامه دوره دارای `/course/` prefix

پرونده `urls.py` از برنامه دوره را ویرایش کنید و الگوهای URL زیر را اضافه کنید:

```
path('subject/<slug:subject>/' ,  
      views.CourseListView.as_view(),  
      name='course_list_subject'),  
  
path('<slug:slug>' ,  
      views.CourseDetailView.as_view(),  
      name='course_detail'),
```

ما الگوهای URL زیر را تعریف می کنیم:

: برای نمایش کلیه دوره های مربوط به یک موضوع course_detail: برای نمایش یک مرور کلی دروس باید قالب های راهنمای استفاده از دوره ایجاد کنید.

ساختار فایل زیر را در templates/courses/directory راهنمای استفاده از دوره ایجاد کنید:

```
course/  
  list.html  
  detail.html
```

الگوی دوره ها / course / list.html را ویرایش کنید و کد زیر را بنویسید:

```
{% extends "base.html" %}  
  
{% block title %}  
  {% if subject %}  
    {{ subject.title }} courses  
  {% else %}
```

```
    All courses
    {% endif %}
    {% endblock %}

    {% block content %}
<h1>
    {% if subject %}
        {{ subject.title }} courses
    {% else %}
        All courses
    {% endif %}
</h1>


<h3>Subjects</h3>
    <ul id="modules">
        <li {% if not subject %}class="selected"{% endif %}>
            <a href="{% url "course_list" %}">All</a>
        </li>
        {% for s in subjects %}
            <li {% if subject == s %}class="selected"{% endif %}>
                <a href="{% url "course_list_subject" s.slug %}">
                    {{ s.title }}
                    <br><span>{{ s.total_courses }} courses</span>
                </a>
            </li>
        {% endfor %}
    </ul>
</div>


{% for course in courses %}

        {% with subject=course.subject %}
            <h3><a href="{% url "course_detail" course.slug %}">
                {{ course.title }}</a></h3>
            <p>
                <a href="{% url "course_list_subject" subject.slug %}">
                    {{ subject }}</a>.
                {{ course.total_modules }} modules.
                Instructor: {{ course.owner.get_full_name }}
            </p>
        {% endwith %}
    {% endfor %}
</div>
    {% endblock %}


```

این الگوی لیست رشته های موجود است. ما یک لیست HTML ایجاد می کنیم تا همه اشیاء Subject را نمایش دهیم و برای هر یک از آنها پیوندی به URL course_list_subject ایجاد کنیم. ما یک کلاس HTML انتخاب شده اضافه می کنیم تا در صورت وجود موضوع فعلی برجسته شود. ما بیش از هر موضوع Course تکرار می کنیم ، تعداد کل مازول ها و نام مربی را نشان می دهیم.

서버 توسعه را اجرا کنید و http://127.0.0.1:8000/ را در مرورگر خود باز کنید. شما باید صفحه ای مشابه صفحه زیر را مشاهده کنید:

The screenshot shows a web application interface for 'EDUCA'. At the top, there is a green header bar with the word 'EDUCA' on the left and 'Sign out' on the right. Below the header, the title 'All courses' is displayed. On the left side, there is a sidebar with a dark background containing a list of subjects and their respective course counts. The subjects listed are 'Subjects' (All), 'Mathematics' (1 COURSES), 'Music' (0 COURSES), 'Physics' (0 COURSES), and 'Programming' (2 COURSES). To the right of the sidebar, the main content area lists courses under each subject. Under 'Subjects', there is one course: 'Django course'. Under 'Mathematics', there is one course: 'Programming. 2 modules. Instructor: Antonio Melé'. Under 'Music', there is one course: 'Python for beginners'. Under 'Physics', there is one course: 'Algebra basics'. Under 'Programming', there are two courses: 'Mathematics. 4 modules. Instructor: Laura Marlon' and another unnamed course.

Subject	Description
Subjects	Django course
All	Programming. 2 modules. Instructor: Antonio Melé
Mathematics	Python for beginners
1 COURSES	Programming. 2 modules. Instructor: Laura Marlon
Music	Algebra basics
0 COURSES	Mathematics. 4 modules. Instructor: Laura Marlon
Physics	
0 COURSES	
Programming	
2 COURSES	

نوار کناری سمت چپ شامل کلیه موضوعات ، از جمله تعداد کل دوره ها برای هر یک از آنها است. برای فیلتر کردن دوره های نمایش داده شده می توانید روی هر موضوعی کلیک کنید.

الگوی دوره ها / course / detail.html را ویرایش کنید و کد زیر را به آن اضافه کنید:

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

{% block title %}

    {{ object.title }}
{% endblock %}

{% block content %}
    {% with subject=course.subject %}
        <h1>
            {{ object.title }}
        </h1>
        <div class="module">
            <h2>Overview</h2>
            <p>
                <a href="{% url "course_list_subject" subject.slug %}">
                    {{ subject.title }}</a>.
                {{ course.modules.count }} modules.
                Instructor: {{ course.owner.get_full_name }}
            </p>
            {{ object.overview|linebreaks }}
        </div>
    {% endwith %}
{% endblock %}
```

در این الگو، یک مرورکلی و جزئیات مربوط به یک دوره واحد را نشان می دهیم. <http://127.0.0.1:8000/> را در مرورگر خود باز کنید و روی یکی از دوره ها کلیک کنید. شما باید صفحه های با ساختار زیر را مشاهده کنید:

Django course

Overview

Programming. 2 modules. Instructor: Antonio Melé

Meet Django. Django is a high-level Python Web framework that encourages rapid development and clean, pragmatic design. Built by experienced developers, it takes care of much of the hassle of Web development, so you can focus on writing your app without needing to reinvent the wheel. It's free and open source.

ما یک مکان عمومی برای نمایش دوره ها ایجاد کرده ایم. در مرحله بعد باید به کاربران اجازه دهیم که به عنوان دانشجو ثبت نام کنند و در دوره ها ثبت نام کنند.

اضافه کردن ثبت نام دانشجویی

با استفاده از دستور زیر یک برنامه جدید ایجاد کنید:

```
python manage.py startapp students
```

پرونده settings.py پروژه آموزشی را ویرایش کنید و برنامه جدید را به تنظیمات INSTALLED_APPS اضافه کنید، به شرح زیر:

```
INSTALLED_APPS = [
    # ...
    'students.apps.StudentsConfig',
]
```

ایجاد نمای ثبت نام دانشجویی

پرونده views.py برنامه دانشجویان را ویرایش کنید و کد زیر را بنویسید:

```
from django.urls import reverse_lazy
from django.views.generic.edit import CreateView
from django.contrib.auth.forms import UserCreationForm
from django.contrib.auth import authenticate, login

class StudentRegistrationView(CreateView):
    template_name = 'students/student/registration.html'
    form_class = UserCreationForm
    success_url = reverse_lazy('student_course_list')

    def form_valid(self, form):
        result = super(StudentRegistrationView,
                      self).form_valid(form)
        cd = form.cleaned_data
        user = authenticate(username=cd['username'],
                            password=cd['password1'])
        login(self.request, user)
        return result
```

این نمای است که به دانشجویان امکان می دهد تا در سایت ما ثبت نام کنند. ما از ژنریک CreatView استفاده می کنیم که عملکرد ایجاد اشیاء مدل را فراهم می کند. این نمای به خصوصیات زیر نیاز دارد: pattern_name: مسیر الگو برای ارائه این نمای.

نام برای ایجاد اشیاء کاربر استفاده می کنیم.

URL_student_course_list: نشانی اینترنتی جهت تغییر مسیر کاربر به هنگام ارسال فرم با موفقیت. ما را معکوس می کنیم ، که می خواهیم در لیست دسترسی به مطالب دوره برای لیست رشته هایی که دانشجویان در آن ثبت نام کرده اند ایجاد کنیم.

هنگامی که داده های فرم معتبر ارسال شده است ، فرم form_valid() اجرا می شود. باید پاسخ HTTP را برگرداند. ما این روش را نادیده می گیریم تا کاربر پس از ورود به سیستم با موفقیت وارد سیستم شوید.

یک پرونده جدید در فهرست برنامه های دانشجویان ایجاد کنید و نام آن را urls.py بگذارید. کد زیر را به آن اضافه کنید:

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

urlpatterns = [
    path('register/',
        views.StudentRegistrationView.as_view(),
        name='student_registration'),
]
```

سپس urls.py اصلی پروژه آموزشی را ویرایش کنید و URL های مربوط به برنامه دانش آموز را با اضافه کردن الگوی زیر به پیکربندی URL خود وارد کنید:

```
urlpatterns = [
    # ...
    path('students/', include('students.urls')),
]
```

ساختم پرونده زیر را در فهرست برنامه های دانشجویی ایجاد کنید:

```
templates/  
    students/  
        student/
```

```
registration.html
```

الگوی `registration.html` را ویرایش کنید و کد زیر را به آن اضافه کنید:

```
{% extends "base.html" %}  
  
{% block title %}  
    Sign up  
{% endblock %}  
  
{% block content %}  
    <h1>  
        Sign up  
    </h1>  
    <div class="module">  
        <p>Enter your details to create an account:</p>  
        <form action="" method="post">  
            {{ form.as_p }}  
            {% csrf_token %}  
            <p><input type="submit" value="Create my account"></p>  
        </form>  
    </div>  
{% endblock %}
```

سرور توسعه را اجرا کنید و <http://127.0.0.1:8000/students/register> را در مرورگر خود باز کنید. باید فرم ثبت نام را به این شکل مشاهده کنید:

Sign up

Enter your details to create an account:

Username: Required. 150 characters or fewer. Letters, digits and @/./+/-/_ only.

Password:

- Your password can't be too similar to your other personal information.
- Your password must contain at least 8 characters.
- Your password can't be a commonly used password.
- Your password can't be entirely numeric.

توجه داشته باشید که نشانی اینترنتی student_course_list مشخص شده در ویژگی sukses_url نمای هنوز وجود ندارد. اگر فرم را ارسال کنید ، Django پس از ثبت نام موفق ، URL را برای تغییر مسیر شما پیدا نمی کند. ما این URL را در دسترسی به بخش مطالب دوره ایجاد خواهیم کرد.

ثبت نام در دوره ها

پس از ایجاد حساب کاربری ، کاربران باید بتوانند در دوره ها ثبت نام کنند.

برای ذخیره ثبت نام ها ، ما باید بین مدل های Course و User یک رابطه بسیار زیاد ایجاد کنیم.

پرونده model.py برنامه دوره را ویرایش کنید و قسمت زیر را به مدل Course اضافه کنید:

```
students = models.ManyToManyField(User,
                                  related_name='courses_joined',
                                  blank=True)
```

از پوسته ، دستور زیر را برای ایجاد تغییر برای این تغییر اجرا کنید:

```
python manage.py makemigrations
```

خروجی مشابه این را مشاهده خواهید کرد:

```
Migrations for 'courses':
courses/migrations/0004_course_students.py
- Add field students to course
```

سپس دستور بعدی را برای اعمال مهاجرت در انتظار اجرا کنید:

```
python manage.py migrate
```

باید خروجی را ببینید که با خط زیر پایان می یابد:

```
Applying courses.0004_course_students... OK
```

اکنون می توانیم دانشجویان را با دوره هایی که در آن ثبت نام کرده اند ، مرتبط کنیم. باید امکاناتی را برای دانش آموزان برای ثبت نام در دوره ها ایجاد کنیم.

یک پرونده جدید در فهرست برنامه های دانشجویان ایجاد کنید و آنرا form.py بنویسید. کد زیر را به آن اضافه کنید

```
from django import forms
from courses.models import Course

class CourseEnrollForm(forms.Form):
    course = forms.ModelChoiceField(queryset=Course.objects.all(),
                                    widget=forms.HiddenInput)
```

ما قصد داریم از این فرم برای دانش آموزان برای ثبت نام در دوره ها استفاده کنیم. قسمت دوره برای دوره هایی است که در آن کاربر ثبت نام می کند.

بنابراین ، این یک ModelChoiceField است. ما از یک ابزارک HiddenInput استفاده می کنیم زیرا نمی خواهیم این قسمت را به کاربر نشان دهیم. ما قصد داریم از این فرم در نمای CourseDetailView استفاده کنیم تا یک دکمه برای ثبت نام نمایش داده شود.

پرونده views.py برنامه دانشجویان را ویرایش کرده و کد زیر را اضافه کنید:

```
from django.views.generic.edit import FormView
from django.contrib.auth.mixins import LoginRequiredMixin
from .forms import CourseEnrollForm

class StudentEnrollCourseView(LoginRequiredMixin, FormView):
    course = None
    form_class = CourseEnrollForm

    def form_valid(self, form):
        self.course = form.cleaned_data['course']
        self.course.students.add(self.request.user)
        return super(StudentEnrollCourseView,
                    self).form_valid(form)

    def get_success_url(self):
        return reverse_lazy('student_course_detail',
                           args=[self.course.id])
```

این StudentEnrollCourseView است. چشم انداز این کار دانش آموزانی را که در دوره ها ثبت نام می کنند ، اداره می کند. نمای از میکسین LoginRequiredMixin به ارث می رسد تا فقط کاربران عضو شده قادر به دسترسی به نمای

باشند. همچنان از نظر Django FormView ارث می برد زیرا ما با ارسال فرم کار می کنیم. ما از فرم CourseEnrollForm برای صفت form_class استفاده می کنیم و همچنان برای ذخیره شیء Course یک ویژگی دوره را تعریف می کنیم. وقتی فرم معتبر است، کاربر فعلی را به دانش آموزانی که در دوره ثبت نام کرده اند اضافه می کنیم.

متده URL () get_success_url را برمی گرداند که کاربر در صورت ارسال موفقیت فرم فرم به آن هدایت می شود. این روش معادل صفت URL_student_course_detail است. ما sukses_url را معکوس می کنیم، که برای نمایش مطالب درسی در بخش دسترسی به دوره مطالب بعدی ایجاد خواهیم کرد.

پرونده urls.py برنامه دانشجویان را ویرایش کنید و الگوی URL زیر را به آن اضافه کنید:

```
path('enroll-course/',
      views.StudentEnrollCourseView.as_view(),
      name='student_enroll_course'),
```

باید فرم دکمه ثبت نام را به صفحه مرور دوره اضافه کنیم. پرونده views.py برنامه دوره را ویرایش کنید و CourseDetailView را اصلاح کنید تا به صورت زیر ظاهر شود:

```
from students.forms import CourseEnrollForm

class CourseDetailView(DetailView):
    model = Course
    template_name = 'courses/course/detail.html'

    def get_context_data(self, **kwargs):
        context = super(CourseDetailView,
                        self).get_context_data(**kwargs)
        context['enroll_form'] = CourseEnrollForm(
            initial={'course':self.object})
        return context
```

ما از روش get_context_data () استفاده می کنیم تا فرم ثبت نام را در متن برای ارائه الگوهای ارائه کنیم. ما قسمت دوره پنهان فرم را با موضوع Course فعلی شروع می کنیم تا مستقیماً ارسال شود.

الگوی دوره ها / course / detail.html را ویرایش کنید و خط زیر را پیدا کنید:

```
 {{ object.overview|linebreaks }}
```

آن را با کد زیر جایگزین کنید:

```
 {{ object.overview|linebreaks }}
{% if request.user.is_authenticated %}
    <form action="{% url "student_enroll_course" %}" method="post">
        {{ enroll_form }}
        {% csrf_token %}
        <input type="submit" class="button" value="Enroll now">
    </form>
{% else %}
    <a href="{% url "student_registration" %}" class="button">
        Register to enroll
    </a>
{% endif %}
```

این دکمه ثبت نام در دوره ها است. در صورت احراز هویت کاربر، دکمه ثبت نام را نشان می دهیم ، از جمله فرم پنهان که به نشانی اینترنتی student_enroll_course اشاره دارد. در صورت عدم احراز هویت کاربر ، ما پیوندی را برای ثبت نام در سیستم عامل نمایش می دهیم.

اطمینان حاصل کنید که سرور توسعه در حال اجرا است ، <http://127.0.0.1:8000> / را در مرورگر خود باز کرده و روی یک دوره کلیک کنید. اگر وارد سیستم شده اید ، باید دکمه ENROLL NOW را که در زیر نمای کلی دوره قرار دارد ، مشاهده کنید به شرح زیر:

Overview

Programming. 2 modules. Instructor: Antonio Melé

Meet Django. Django is a high-level Python Web framework that encourages rapid development and clean, pragmatic design. Built by experienced developers, it takes care of much of the hassle of Web development, so you can focus on writing your app without needing to reinvent the wheel. It's free and open source.

ENROLL NOW

اگر وارد سیستم نشده باشید ، به جای آن دکمه REGISTER TO ENROLL را مشاهده خواهید کرد.

دسترسی به مطالب دوره

برای نمایش دوره هایی که دانش آموزان در آن ثبت نام می کنند ، به یک نمایش نیاز داریم و یک نمای برای دسترسی به مطالب واقعی دوره. پرونده views.py برنامه دانشجویان را ویرایش کنید و کد زیر را به آن اضافه کنید:

```
from django.views.generic.list import ListView
from courses.models import Course

class StudentCourseListView(LoginRequiredMixin, ListView):
    model = Course
    template_name = 'students/course/list.html'

    def get_queryset(self):
        qs = super(StudentCourseListView, self).get_queryset()
        return qs.filter(students__in=[self.request.user])
```

این دیدگاه برای دانشجویان است که دوره هایی را که در آن ثبت نام کرده اند ، لیست کنند.

برای اطمینان از اینکه فقط کاربران عضو شده قادر به دسترسی به نمای هستند ، از LoginRequiredMixin به ارث می برند. همچنین از نمایش عمومی ListView برای نمایش لیستی از اشیاء Course به ارث می برد. ما برای بازیابی تنها دوره هایی که کاربر در آن ثبت نام کرده است ، متد get_queryset () را رد می کنیم. ما برای انجام این کار QuerySet را توسط قسمت ManyToManyField دانشجویی فیلتر می کنیم. سپس کد زیر را به پرونده views.py اضافه کنید:

```
from django.views.generic.detail import DetailView

class StudentCourseDetailView(DetailView):
    model = Course
    template_name = 'students/course/detail.html'

    def get_queryset(self):
        qs = super(StudentCourseDetailView, self).get_queryset()
        return qs.filter(students__in=[self.request.user])

    def get_context_data(self, **kwargs):

        context = super(StudentCourseDetailView,
                        self).get_context_data(**kwargs)
        # get course object
        course = self.get_object()
        if 'module_id' in self.kwargs:
            # get current module
            context['module'] = course.modules.get(
                id=self.kwargs['module_id'])
        else:
            # get first module
            context['module'] = course.modules.all()[0]
        return context
```

این StudentCourseDetailView است. ما از روش get_queryset () را رد می کنیم تا پایه QuerySet را در دوره هایی که کاربر در آن ثبت نام کرده محدود کنیم. در صورت دادن پارامتر __URL_module ، ما همچنین متد get_context_data را معرفی کنیم.

) را نادیده می‌گیریم تا مژول دوره را در متن تنظیم کنیم. در غیر این صورت، مژول اول دوره را تنظیم می‌کنیم. به این ترتیب، دانش آموزان قادر خواهند بود از طریق مژول‌ها در یک دوره حرکت کنند.

پرونده urls.py برنامه دانشجویان را ویرایش کنید و الگوهای URL زیر را به آن اضافه کنید:

```
path('courses/',
      views.StudentCourseListView.as_view(),
      name='student_course_list'),  
  
path('course/<pk>',
      views.StudentCourseDetailView.as_view(),
      name='student_course_detail'),  
  
path('course/<pk>/<module_id>',
      views.StudentCourseDetailView.as_view(),
      name='student_course_detail_module'),
```

ساختار فایل زیر را در templates/students/ فهرست راهنمای برنامه دانش آموزان ایجاد کنید:

```
course/
  detail.html
  list.html
```

الگوی students/course/list.html را ویرایش کنید و کد زیر را به آن اضافه کنید:

```

{% extends "base.html" %}

{% block title %}My courses{% endblock %}

{% block content %}
    <h1>My courses</h1>

    <div class="module">
        {% for course in object_list %}
            <div class="course-info">
                <h3>{{ course.title }}</h3>
                <p><a href="{% url "student_course_detail" course.id %}">
                    Access contents</a></p>
            </div>
        {% empty %}
        <p>
            You are not enrolled in any courses yet.
            <a href="{% url "course_list" %}">Browse courses</a>
            to enroll in a course.
        </p>
    {% endfor %}
    </div>
{% endblock %}

```

در این الگو دوره هایی که کاربر در آن ثبت نام کرده است را نشان می دهد.

به یاد داشته باشید وقتی دانش آموز جدید با موفقیت در سیستم عامل ثبت نام می کند ، آنها به URL student_course_list هدایت می شوند. باید دانشجویان را نیز هنگام ورود به سیستم عامل ، به این URL هدایت کنیم.

پرونده settings.py پروژه آموزشی را ویرایش کنید و کد زیر را به آن اضافه کنید:

```

from django.urls import reverse_lazy
LOGIN_REDIRECT_URL = reverse_lazy('student_course_list')

```

در صورتی که هیچ پارامتر بعدی در درخواست وجود نداشته باشد ، این تنظیمات است که توسط ماژول auth برای هدایت کاربر به سمت ورود به سیستم موفق می شود. پس از ورود به سیستم موفقیت آمیز ، دانشجویان برای مشاهده دوره هایی که در آن ثبت نام کرده اند به آدرس URL student_course_list هدایت می شوند.

الگوی students/course/detail.html را ویرایش کنید و کد زیر را به آن اضافه کنید:

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

{% block title %}
    {{ object.title }}
{% endblock %}

{% block content %}
    <h1>
        {{ module.title }}
    </h1>
    <div class="contents">
        <h3>Modules</h3>
        <ul id="modules">
            {% for m in object.modules.all %}
                <li data-id="{{ m.id }}" {% if m == module
                %}class="selected"
                {% endif %}>
                    <a href="{% url "student_course_detail_module"
                    object.id m.id %}">
                        <span>
                            Module <span class="order">{{ m.order|add:1 }}</span>
                        </span>
                        <br>
                        {{ m.title }}
                    </a>
                </li>
            {% endfor %}
        </ul>
    </div>
</div>
```

```

    {% empty %}
        <li>No modules yet.</li>
    {% endfor %}
    </ul>
</div>
<div class="module">
    {% for content in module.contents.all %}
        {% with item=content.item %}
            <h2>{{ item.title }}</h2>
            {{ item.render }}
        {% endwith %}
    {% endfor %}
</div>
{% endblock %}

```

این الگوی برای دانشجویان ثبت نام شده برای دسترسی به مطالب یک دوره است. ابتدا ، ما یک لیست HTML ایجاد می کنیم که شامل تمام مژول های دوره و برجسته کردن مژول فعلی است. سپس ، بیش از محتوا مژول فعلی تکرار می کنیم و به هر مورد محتوا دسترسی پیدا می کنیم تا با استفاده از {{ item.render }} نمایش داده شود. می خواهیم روش رندر () را به مدل های بعدی اضافه کنیم. با استفاده از این روش می توان به درستی مطالب را رندر کرد.

ارائه انواع مختلف محتوا

ما باید راهی برای ارائه هر نوع محتوا ارائه دهیم. پرونده model.py برنامه دوره را ویرایش کنید و render() زیر را به مدل itemBase اضافه کنید:

```

from django.template.loader import render_to_string
from django.utils.safestring import mark_safe

class ItemBase(models.Model):
    # ...

    def render(self):
        return render_to_string('courses/content/{}.html'.format(
            self._meta.model_name), {'item': self})

```

در این روش از عملکرد render_to_string() برای ارائه یک الگو و بازگرداندن محتواهای ارائه شده به عنوان یک رشته استفاده می‌شود. هر نوع محتوا با استفاده از الگویی به نام مدل محتوا ارائه می‌شود. ما از self._meta.model_name استفاده می‌کنیم تا نام الگوی مناسب برای هر مدل محتوا را بصورت پویا تولید کنیم. (یک رابط مشترک برای ارائه مطالب متنوع ارائه می‌دهد.

ساختار فایل زیر را در قالب templates/courses/راهنمای برنامه دوره ایجاد کنید:

```
content/
  text.html
  file.html
  image.html
  video.html
```

قالب courses/content/text.html را ویرایش کنید و این کد را بنویسید:

```
  {{ item.content|linebreaks|safe }}
```

الگوی courses/content/file.html را ویرایش کنید و موارد زیر را اضافه کنید:

```
<p><a href="{{ item.file.url }}" class="button">Download file</a></p>
```

الگوی courses/content/image.html را ویرایش کنید و بنویسید:

```
<p></p>
```

برای کارکردن پرونده‌هایی که با FileField و ImageField بارگذاری شده‌اند، باید پروژه خود را برای ارائه فایل‌های رسانه‌ای با سرور توسعه تنظیم کنیم.

پرونده settings.py پروژه خود را ویرایش کنید و کد زیر را به آن اضافه کنید:

```
MEDIA_URL = '/media/'  
MEDIA_ROOT = os.path.join(BASE_DIR, 'media/')
```

به یاد داشته باشید که MEDIA_URL اصلی برای ارائه پرونده های رسانه ای بارگذاری شده است و MEDIA_ROOT مسیر محلی است که پرونده ها در آن قرار دارند.

پرونده اصلی urls.py پروژه خود را ویرایش کنید و واردات زیر را اضافه کنید:

```
from django.conf import settings  
from django.conf.urls.static import static
```

سپس خطوط زیر را در انتهای پرونده بنویسید:

```
if settings.DEBUG:  
    urlpatterns += static(settings.MEDIA_URL,  
                          document_root=settings.MEDIA_ROOT)
```

اکنون پروژه شما آماده بارگذاری و سرویس پرونده های رسانه ای است. سرور توسعه Django وظیفه ارائه پرونده های رسانه ای را در هنگام توسعه (به عنوان مثال ، هنگامی که تنظیم DEBUG روی True تنظیم شده است) خواهد بود. به یاد داشته باشید که سرور توسعه برای استفاده در تولید مناسب نیست. شما یاد می گیرید که چگونه یک بخش تولید را در بخش Going Live ، 13 تنظیم کنید.

همچنین باید الگویی برای ردیابی اشیاء ویدیویی ایجاد کنیم. ما برای تعییه محتوای ویدئویی از فیلم Django-embed-video استفاده خواهیم کرد. Django-embed-video یک برنامه شخص ثالث Django است که به شما امکان می دهد فیلم ها را در قالب های خود ، از منابع مانند YouTube یا Vimeo ، با قرار دادن آدرس اینترنتی عمومی فیلم جاسازی کنید.

بسته را با دستور زیر نصب کنید:

```
pip install django-embed-video
```

پرونده تنظیمات.py پروژه خود را ویرایش کنید و برنامه را به INSTALLED_APPS اضافه کنید ، تنظیم کنید به شرح زیر:

```
INSTALLED_APPS = [
    # ...
    'embed_video',
]
```

می توانید مستندات برنامه django-embed-video را در <https://django-embed-video.readthedocs.io/fa/latest/video.html> پیدا کنید.

الگوی <https://courses.djangogirls.org/courses/content/video.html> را ویرایش کنید و موارد زیر را بنویسید کد:

```
{% load embed_video_tags %}
{% video item.url "small" %}
```

اکنون سرور توسعه را اجرا کنید و به مرورگر خود به <http://127.0.0.1:8000/course/mine> دسترسی پیدا کنید. با کاربری که به گروه مربیان تعلق دارد ، به سایت دسترسی پیدا کنید و مطالب مختلف را به یک دوره اضافه کنید. برای گنجاندن محتوای ویدیو ، فقط می توانید URL های YouTube مانند <https://www.youtube.com/watch?v=bgV39DImZ2U> را کپی کنید؟

و آن را در قسمت url فرم وارد کنید.

پس از افزودن محتوا به دوره باز <http://127.0.0.1:8000/> ، بر روی دوره کلیک کنید و بر روی دکمه ENROLL NOW کلیک کنید. شما باید در دوره ثبت نام کرد و به URL `student_course_detail` هدایت شوید.

تصویر زیر محتوای دوره نمونه را نشان می دهد:

Introduction to Django

Modules

MODULE 1
Introduction to Django

MODULE 2
Configuring Django

MODULE 3
Your first Django project

MODULE 4
Django URLs

Why Django?

Meet Django. Django is a high-level Python Web framework that encourages rapid development and clean, pragmatic design. Built by experienced developers , it takes care of much of the hassle of Web development, so you can focus on writing your app without needing to reinvent the wheel. It's free and open source.

Django video



عالی! شما یک رابط مشترک برای ارائه انواع مختلف محتوای دوره ایجاد کرده اید.

با استفاده از چارچوب کش

درخواست های HTTP به برنامه وب شما معمولاً مستلزم دسترسی به پایگاه داده ، پردازش داده ها و ارائه قالب است. این از نظر پردازش بسیار گرانتر از سرویس دهی به وب سایت استاتیک است.

هنگام شروع سایت بیشتر و بیشتر برای ترافیک ، سایت برخی از درخواست ها قابل توجه است. اینجاست که ذخیره سازی گران می شود. با ذخیره کردن نمایش داده شد ، نتایج محاسبه یا محتوای ارائه شده در یک درخواست HTTP ، از هزینه های گران قیمت در درخواست های زیر جلوگیری می کنید. این ترجمه به زمان پاسخگویی کوتاهتر و پردازش کمتری در سمت سرور تبدیل می شود.

Django شامل یک سیستم کش قوی است که به شما امکان می دهد داده های ذخیره شده با سطوح مختلف گرانول را ذخیره کنید. می توانید یک پرس و جو واحد ، خروجی یک نمای خاص ، بخش هایی از محتوای قالب ارائه شده یا کل سایت خود را ذخیره کنید. موارد برای مدت زمان پیش فرض در سیستم حافظه پنهان ذخیره می شوند. می توانید زمان پیش فرض داده های ذخیره شده را تعیین کنید.

به این صورت است که معمولاً هنگام درخواست درخواست HTTP از چارچوب کش استفاده خواهد کرد:

1. سعی کنید داده های درخواستی را در حافظه پنهان پیدا کنید

2. در صورت یافتن ، داده های ذخیره شده را برگردانید

3. در صورت یافت نشد ، مراحل زیر را انجام دهید:

1. پرس و جو یا پردازش مورد نیاز برای به دست آوردن داده را انجام دهید

2. داده های تولید شده را در حافظه نهان ذخیره کنید

3. داده ها را برگردانید

می توانید اطلاعات دقیق در مورد سیستم حافظه پنهان جنگور را در اینجا بخوانید

<http://docs.djangoproject.com/fa/2.0/topics/cache>

پس زمینه های حافظه نهان موجود

جنگو با چندین بورد حافظه پنهان همراه است.

این موارد زیر است:

یک پس زمینه backends.memcached.PyLibMCCache یا backends.memcached.MemcachedCache یک سرور حافظه مبتنی بر حافظه سریع و کارآمد است. باطن مورد استفاده بستگی به اتصالات Memcached Python شما انتخاب می کند.

از بانک اطلاعاتی به عنوان سیستم کش استفاده کنید.

از سیستم ذخیره فایل استفاده کنید.

هر مقدار حافظه پنهان را به عنوان یک پرونده جداگانه سریال سازی و ذخیره می کند.

باطن حافظه محلی. این باطن حافظه پیشفرض است.

باطن کش ساختگی که فقط برای توسعه در نظر گرفته شده است. این رابط حافظه پنهان را بدون ذخیره کردن هر چیزی پیاده سازی می کند. این حافظه پنهان برای هر فرآیند و بدون امنیت از نخ استفاده می شود.

نصب Memcached

ما قصد داریم از Backend Memcached در حافظه اجرایی شود و به مقدار مشخصی از RAM اختصاص داده می شود. هنگامی که RAM اختصاص داده شده پر است ، Memcached شروع به برداشتن قدیمی ترین داده ها برای ذخیره اطلاعات جدید می کند.

Memcached را از https://memcached.org/downloads بارگیری کنید. اگر از لینوکس استفاده می کنید ، می توانید Memcached را با استفاده از دستور زیر نصب کنید:

```
./configure && make && make test && sudo make install
```

اگر از macOS X استفاده می کنید ، می توانید Memcached را با مدیر بسته Homebrew با استفاده از دستور brew install Memcached نصب کنید. می توانید Homebrew را از https://brew.sh/ بارگیری کنید.

پس از نصب Memcached ، یک پوسته را باز کرده و با استفاده از دستور زیر آن را شروع کنید:

```
memcached -l 127.0.0.1:11211
```

Memcached به طور پیش فرض روی پورت 11211 اجرا خواهد شد. با این وجود می توانید با استفاده از گزینه -l یک هاست و پورت دلخواه را تعیین کنید. می توانید اطلاعات بیشتر در مورد Memcached را در <https://memcached.org> پیدا کنید.

پس از نصب Python ، باید اتصالات Memcached آن را نصب کنید.

می توانید با دستور زیر این کار را انجام دهید:

```
pip install python-memcached==1.59
```

تنظیمات حافظه نهان

Django تنظیمات حافظه نهان زیر را ارائه می دهد:

CACHES: فرهنگ لغت حاوی کلیه حافظه پنهان های موجود برای این پروژه.

CACHE_MIDDLEWARE_ALIAS: مخفیگاه حافظه پنهان برای ذخیره سازی.

CACHE_MIDDLEWARE_KEY_PREFIX: پیشوند مورد استفاده برای کلیدهای حافظه نهان.

اگر مشترک حافظه پنهان بین چندین سایت باشد ، پیشوندی تنظیم کنید تا از برخورد کلیدی جلوگیری کنید.

CACHE_MIDDLEWARE_SECONDS: تعداد پیش فرض ثانیه برای ذخیره شدن صفحات.

سیستم حافظه پنهان پروژه می تواند با استفاده از تنظیمات CACHES پیکربندی شود. این تنظیم یک فرهنگ لغت است که به شما امکان می دهد پیکربندی چندین حافظه پنهان را مشخص کنید. هر حافظه پنهان موجود در فرهنگ لغت cache می تواند داده های زیر را مشخص کند:

سابقه و هدف: حافظه نهان ذخیره شده برای استفاده.

KEY_FUNCTION: یک رشته حاوی یک مسیر نقطه ای به یک فراخوانی که یک پیشوند ، نسخه و کلید را به عنوان آرگومان می گیرد و یک کلید کش نهایی را برمی گرداند.

KEY_PREFIX: یک پیشوند رشته برای همه کلیدهای حافظه نهان ، برای جلوگیری از برخورد.

مکان: محل حافظه پنهان. بسته به باطن حافظه نهان ، این ممکن است یک دایرکتوری ، میزبان و پورت یا نامی برای باطن حافظه باشد.

گزینه ها: هر پارامتر اضافی که باید به باطن حافظه پنهان منتقل شود.

TIMEOUT: مدت زمان پیش فرض ، برای چند ثانیه برای ذخیره سازی کلیدهای حافظه نهان ، ذخیره می شود. 300 ثانیه به طور پیش فرض ، که پنج دقیقه است. اگر روی None تنظیم شود ، کلیدهای حافظه پنهان منقضی نمی شوند.

VERSION: شماره نسخه پیش فرض برای کلیدهای حافظه نهان است. برای نسخه نویسی حافظه نهان مفید است.

افزودن Memcached به پروژه شما

بایاید حافظه نهان را برای پروژه خود پیکربندی کنیم. پرونده settings.py پروژه آموزشی را ویرایش کنید و کد زیر را به آن اضافه کنید:

```
CACHES = {
    'default': {
        'BACKEND': 'django.core.cache.backends.memcached.MemcachedCache',
        'LOCATION': '127.0.0.1:11211',
    }
}
```

ما از باطن MemcachedCache استفاده می کنیم. ما مکان آن را با استفاده از آدرس مشخص می کنیم: notation port . می توانید از لیستی برای LOCATION استفاده کنید. اگر چندین مورد Memcached دارید ،

ناظارت بر Memcached

برای ناظارت بر Memcached ، از یک بسته شخص ثالث به نام django-memcache-status استفاده خواهیم کرد. این برنامه آماری را برای نمونه های Memcached شما در سایت مدیریت نمایش می دهد. آن را با دستور زیر نصب کنید:

```
pip install django-memcache-status==1.3
```

پرونده تنظیمات را ویرایش کنید و 'memcache_status' را به تنظیمات INSTALLED_APPS اضافه کنید:

```
INSTALLED_APPS = [
    # ...
    'memcache_status',
]
```

اطمینان حاصل کنید که Memcached در حال اجراست ، سرور توسعه را در یک پنجره پوسته دیگر شروع کنید و /http://127.0.0.1:8000/admin را در مرورگر خود باز کنید. با استفاده از یک superuser به سایت مدیریت وارد شوید. باید بلوک زیر را مشاهده کنید:

MEMCACHED: DEFAULT: 127.0.0.1:11211 (1) - 0% LOAD

این نمودار استفاده از حافظه نهان را نشان می دهد. رنگ سبز نشانگر حافظه نهان رایگان است در حالی که قرمز فضای استفاده شده را نشان می دهد. اگر روی عنوان کادر کلیک کنید ،

این آمار دقیق از نمونه Memcached شما را نشان می دهد.

ما Memcached را برای پروژه خود تنظیم کرده ایم و قادر به نظارت بر آن هستیم. باید شروع به ذخیره اطلاعات کنیم!

سطح حافظه نهان

جنگو سطوح زیر ذخیره شده را که به ترتیب صعودی ذرات در اینجا ذکر شده است ، فراهم می کند:

API نهانگاه سطح پایین: بالاترین میزان دانه را فراهم می کند.

به شما امکان می دهد سوالات یا محاسبات خاص را ذخیره کنید.

حافظه پنهان برای هر بازدید: حافظه پنهان را برای نمایش های فردی فراهم می کند.

حافظه نهان الگو: به شما امکان می دهد قطعات قالب را ذخیره کنید.

حافظه پنهان در هر سایت: حافظه نهان بالاترین سطح. این کل سایت شما را ذخیره می کند.

استفاده از API حافظه نهان سطح پایین

API حافظه پنهان سطح پایین به شما امکان می دهد تا اشیاء را در حافظه نهان خود ذخیره کنید. در django.core.cache واقع شده است. شما می توانید آن را مانند این وارد کنید:

```
from django.core.cache import cache
```

با استفاده از حافظه پنهان پیش فرض. معادل حافظه پنهان است ["پیش فرض"].

دسترسی به یک حافظه پنهان خاص از طریق نام مستعار آن نیز امکان پذیر است:

```
from django.core.cache import caches  
my_cache = caches['alias']
```

بایاید نگاهی به نحوه عملکرد حافظه پنهان API بیندازیم. پوسته را با فرمان python manage.py باز کرده و کد زیر را اجرا کنید:

```
>>> from django.core.cache import cache  
>>> cache.set('musician', 'Django Reinhardt', 20)
```

ما برای ذخیره یک کلید به نام "نوازنده" با مقداری که رشتہ "Django Reinhardt" به مدت 20 ثانیه باشد ، به باطن حافظه پنهان پیشفرض و استفاده از مجموعه (کلید ، مقدار ، زمان بندی) می پردازیم. اگر ما یک زمان بندی را مشخص نکنیم ، از تنظیم پیش فرض مشخص شده برای باطن حافظه پنهان در تنظیمات CACHES استفاده می کند. اکنون کد زیر را اجرا کنید:

```
>>> cache.get('musician')  
'Django Reinhardt'
```

ما کلید را از حافظه پنهان بازیابی می کنیم. 20 ثانیه صبر کنید و همان کد را اجرا کنید:

```
>>> cache.get('musician')
```

این بار هیچ ارزشی ارسال نمی شود. کلید کش "نوازنده" منقضی شده است و روش دریافت () هیچکدام را بر می گرداند زیرا دیگر کلید در حافظه پنهان نیست.

بایاید QuerySet را با کد زیر ذخیره کنیم:

```
>>> from courses.models import Subject  
>>> subjects = Subject.objects.all()  
>>> cache.set('all_subjects', subjects)
```

ما یک QuerySet را در مدل Subject انجام می دهیم و اشیاء برگشتی را در کلید 'all_subjects' ذخیره می کنیم. باید داده های ذخیره شده را بازیابی کنیم:

```
>>> cache.get('all_subjects')  
<QuerySet [<Subject: Mathematics>, <Subject: Music>, <Subject: Physics>,  
<Subject: Programming>]>
```

ما در نظر داریم برخی از نمایش داده شد. پرونده views.py برنامه دوره ها را ویرایش کنید و واردات زیر را اضافه کنید:

```
from django.core.cache import cache
```

در متده get () CourseListView خط زیر را جایگزین کنید:

```
subjects = Subject.objects.annotate(  
    total_courses=Count('courses'))
```

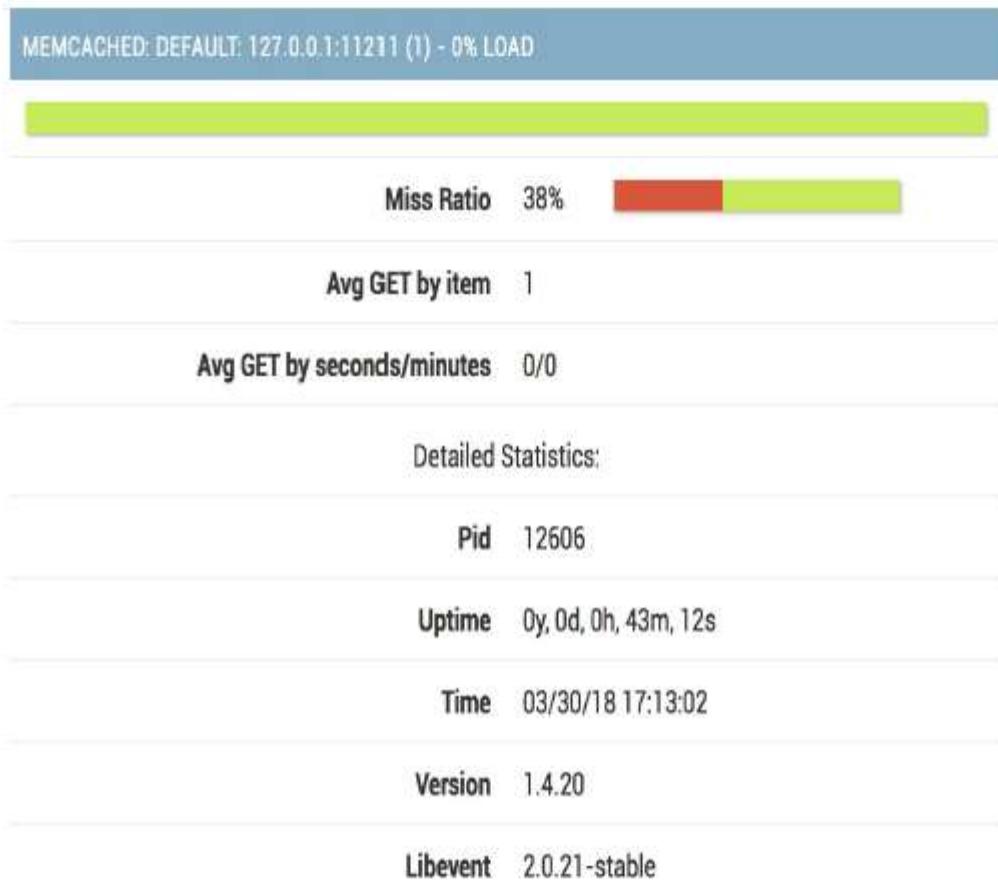
آن را با موارد زیر جایگزین کنید:

```
subjects = cache.get('all_subjects')
if not subjects:
    subjects = Subject.objects.annotate(
```

```
        total_courses=Count('courses'))
cache.set('all_subjects', subjects)
```

در این کد ، ما سعی می کنیم با استفاده از `cache.get()` کلید `all_students` را از حافظه نهانگاه استفاده کنیم. در صورت یافتن کلید داده شده ، این برگردانده می شود. اگر هیچ کلید یافت نشد (هنوز ذخیره نشده یا ذخیره نشده است ، اما به پایان رسیده است) ، ما برای جستجوی همه اشیاء `Subject` و تعداد دوره های آنها ، `query` را انجام دادیم و نتیجه را با استفاده از `cache.set()` ذخیره می کنیم.

سرور توسعه را اجرا کنید و `http://127.0.0.1:8000/` را در مرورگر خود باز کنید. وقتی نمای اجرا شد ، کلید حافظه پنهان یافت نمی شود و `QuerySet` اجرا می شود. `http://127.0.0.1:8000/admin/` را در مرورگر خود باز کنید و آمار راگسترش دهید. باید داده های استفاده برای حافظه پنهان را مانند صفحه زیر مشاهده کنید:



به موارد Curr نگاهی بیندازید ، که باید 1 باشد. این نشان می دهد که یک مورد در حال حاضر در حافظه پنهان ذخیره شده است. دریافت بازدیدها نشان می دهد که تعداد بسیاری از دستورات موفق بودند و دریافت Miss ها درخواست های دریافت کلیدهای را که گم شده اند نشان می دهد. Miss Ratio با استفاده از هر دو محاسبه می شود.

اکنون با استفاده از مرورگر خود به <http://127.0.0.1:8000> بروید و صفحه را بارها بارگیری کنید. اگر اکنون به آمار حافظه پنهان نگاهی بیندازید ، چندین مورد دیگر خوانده شده (دریافت بازدیدها و Cmd دریافت افزایش خواهد یافت).

ذخیره سازی بر اساس داده های پویا

بسیاری اوقات می خواهید چیزی را که مبتنی بر داده های پویا است ، ذخیره کنید. در این موارد ، شما باید کلیدهای دینامیک بسازید که شامل تمام اطلاعات مورد نیاز برای شناسایی یکپارچه داده های ذخیره شده ذخیره شده است. پرونده views.py برنامه دوره را ویرایش کنید و نمای CourseListView را اصلاح کنید تا اینگونه به نظر برسد:

```

class CourseListView(TemplateResponseMixin, View):
    model = Course
    template_name = 'courses/course/list.html'

    def get(self, request, subject=None):
        subjects = cache.get('all_subjects')
        if not subjects:
            subjects = Subject.objects.annotate(
                total_courses=Count('courses'))
            cache.set('all_subjects', subjects)
        all_courses = Course.objects.annotate(
            total_modules=Count('modules'))
        if subject:
            subject = get_object_or_404(Subject, slug=subject)
            key = 'subject_{}_.courses'.format(subject.id)
            courses = cache.get(key)
            if not courses:
                courses = all_courses.filter(subject=subject)
                cache.set(key, courses)
        else:
            courses = cache.get('all_courses')
            if not courses:
                courses = all_courses
                cache.set('all_courses', courses)
        return self.render_to_response({'subjects': subjects,
                                        'subject': subject,
                                        'courses': courses})

```

در این حالت ، ما هم کلیه دوره ها و هم دوره های فیلتر شده توسط موضوع را ذخیره می کنیم. در صورت عدم ارائه موضوع ، از کلید حافظه نهانگاه `_all_courses` برای ذخیره کلیه دوره ها استفاده می کنیم. اگر موضوعی وجود داشته باشد ، ما کلید را بصورت پویا با "موضوع _ {} _ دوره " (موضوع.id) می سازیم.

توجه به این نکته مهم است که شما نمی توانید از `QuerySet` ذخیره شده برای ساخت سایر `QuerySets` استفاده کنید ، زیرا آنچه که در آن ذخیره کردید در واقع نتایج `QuerySet` است. بنابراین شما نمی توانید موارد زیر را انجام دهید:

```

courses = cache.get('all_courses')
courses.filter(subject=subject)

```

در عوض ، شما باید پایه total_modules = تعداد ('ماژول ها') QuerySet Course.objects.annotate را ایجاد کنید ، که تا زمانی که مجبور نشود اجرا نخواهد شد و از آن برای محدود کردن بیشتر QuerySet با all_courses.filter استفاده کنید (موضوع = موضوع در صورت یافتن داده در حافظه نهان .

قطعات قالب ذخیره

ذخیره قالب قطعات یک رویکرد سطح بالاتر است. شما باید با استفاده از {load cache %} ، برچسب های قالب کش را در الگوی خود بارگیری کنید.

سپس می توانید از قالب {cache %} برای ذخیره کردن قطعات قالب خاص استفاده کنید. شما معمولاً از برچسب قالب به شرح زیر استفاده خواهید کرد:

```
{% cache 300 fragment_name %}
...
{% endcache %}
```

برچسب {cache %} دو آرگومان مورد نیاز دارد: زمان پایان ، به مدت چند ثانیه و نامی برای قطعه. اگر بسته به داده های پویا نیاز به حافظه پنهان دارید ، می توانید با ارسال آرگومان های اضافی به برچسب قالب {cache %} برای شناسایی منحصر به فرد این قطعه ، این کار را انجام دهید.

برنامه دانشجویان را ویرایش کنید. درست پس از تگ {extends %} ، کد زیر را در قسمت بالای آن اضافه کنید:

```
{% load cache %}
```

سپس خطوط زیر را جایگزین کنید:

```
{% for content in module.contents.all %}  
  {% with item=content.item %}  
    <h2>{{ item.title }}</h2>  
    {{ item.render }}  
  {% endwith %}  
{% endfor %}
```

آنها را با موارد زیر جایگزین کنید:

```
{% cache 600 module_contents module %}  
  {% for content in module.contents.all %}  
    {% with item=content.item %}  
      <h2>{{ item.title }}</h2>  
      {{ item.render }}  
    {% endwith %}  
  {% endfor %}  
{% endcache %}
```

ما با استفاده از نام `module_contents` و انتقال جسم فعلی `Module` به آن ، این قطعه قالب را ذخیره می کنیم. بنابراین ، ما به طور منحصر به فرد قطعه را شناسایی می کنیم. این مهم است برای جلوگیری از گرفتن مطالب یک مازول و ارائه مطالب اشتباه هنگام درخواست یک مازول متفاوت.

ذخیره نماها

می توانید با استفاده از دکوراتور `cache_page` واقع در `django.views.decorators.cache` ، خروجی نماهای فردی را ذخیره کنید. دکوراتور به یک استدلال زمانبندی (در چند ثانیه) نیاز دارد.

باید از آن در نظرات خود استفاده کنیم. پرونده `urls.py` برنامه دانشجویان را ویرایش کنید و واردات زیر را اضافه کنید:

```
from django.views.decorators.cache import cache_page
```

سپس دکوراتور `cache_page` را روی الگوهای URL `student_course_detail` و `student_course_detail_module` قرار دهید ، به شرح زیر:

```
path('course/<pk>/',  
      cache_page(60 * 15)(views.StudentCourseDetailView.as_view()),  
      name='student_course_detail'),  
  
path('course/<pk>/<module_id>/',  
      cache_page(60 * 15)(views.StudentCourseDetailView.as_view()),  
      name='student_course_detail_module'),
```

اکنون ، نتیجه‌ی `StudentCourseDetailView` به مدت 15 دقیقه ذخیره می‌شود.

استفاده از حافظه نهانگاه در هر سایت

این حافظه نهان بالاترین سطح است. به شما امکان می‌دهد کل سایت خود را ذخیره کنید.

برای اجازه به حافظه پنهان در هر سایت ، فایل تنظیمات `.py` پروژه خود را ویرایش کنید و کلاس‌های `MIDDLEWARE` را به تنظیمات `FetchFromCacheMiddleware` و `UpdateCacheMiddleware` اضافه کنید ، به شرح زیر:

```
MIDDLEWARE = [  
    'django.middleware.security.SecurityMiddleware',  
    'django.contrib.sessions.middleware.SessionMiddleware',  
    'django.middleware.cache.UpdateCacheMiddleware',  
    'django.middleware.common.CommonMiddleware',  
    'django.middleware.cache.FetchFromCacheMiddleware',  
    # ...  
]
```

به یاد داشته باشید که وسائل میانی در مرحله درخواست به ترتیب معین و در مرحله پاسخ به ترتیب معکوس اجرا می‌شوند.

قرار داده شده است زیرا در زمان پاسخ وقتی اجرا می شود که واسطه به ترتیب معکوس اجرا شود. عمدتاً بعد از CommonMiddleware FetchFromCacheMiddleware قرار می گیرد زیرا نیاز به دسترسی به داده های درخواست تنظیم شده توسط دومی دارد.

سپس تنظیمات زیر را به پرونده تنظیمات اضافه کنید:

```
CACHE_MIDDLEWARE_ALIAS = 'default'  
CACHE_MIDDLEWARE_SECONDS = 60 * 15 # 15 minutes  
CACHE_MIDDLEWARE_KEY_PREFIX = 'educa'
```

در این تنظیمات ، از حافظه پنهان پیش فرض برای میان افزارکش خود استفاده می کنیم و زمان جهانی حافظه پنهان را 15 دقیقه تنظیم می کنیم. ما همچنین یک پیشوند برای همه کلیدهای حافظه نهان را تعیین می کنیم تا در صورت استفاده از باطن میکسان برای چندین پروژه ، از تصادم جلوگیری کنیم. اکنون سایت ما برای همه درخواستهای GET محتوای Memcached ذخیره شده را ذخیره کرده و برمی گرداند.

ما این کار را کرده ایم تا عملکرد حافظه نهان در هر سایت را آزمایش کنیم. با این حال ، حافظه نهان هر سایت برای ما مناسب نیست ، زیرا نمایه های مدیریت دوره باید داده های به روز رانشان دهنند تا فوراً منعکس کننده هرگونه تغییر باشد. بهترین روش برای دنبال کردن در پروژه ما ، نادیده گرفتن الگوهای یا نماهایی است که برای نمایش محتوای دوره به دانشجویان استفاده می شود.

ما یک مرور کلی از روش های ارائه شده توسط جنگو برای ذخیره داده های داده شده مشاهده کرده ایم. شما باید استراتژی حافظه پنهانی خود را عاقلانه تعریف کنید و گرانترین QuerySets یا محاسبات را در اولویت قرار دهید.

خلاصه

در این فصل دیدگاه های عمومی را برای دوره ها ایجاد کردیم و شما سیستمی برای ثبت نام و ثبت نام در دوره ها ایجاد کرده اید.

ما Memcached را نصب کردیم و سطح حافظه پنهان مختلف را اجرا کردیم.

در فصل بعد ، یک API RESTful برای پروژه شما ایجاد خواهیم کرد.

فصل دوازدهم

ساختن یک API

در فصل قبل شما سیستم ثبت نام و ثبت نام دانش آموزان در دوره ها را ایجاد کردید. شما برای نمایش مطالب درسی دیدگاه ایجاد کرده اید و نحوه استفاده از چارچوب حافظه پنهان Django را یاد گرفته اید. در این فصل نحوه انجام موارد زیر را یاد خواهید گرفت:

یک API RESTful بسازید

احراز هویت و مجوزها برای نمایش API

مجموعه ها و روترهای نمای API ایجاد کنید

ساختن یک API RESTful

ممکن است بخواهید یک رابط برای سایر سرویس‌ها ایجاد کنید تا با برنامه وب خود ارتباط برقرار کنید. با ساختن یک API، می‌توانید به اشخاص ثالث اجازه دهید تا اطلاعات را مصرف کنند و با برنامه خود به طور برنامه‌ای کار کنند.

چندین روش وجود دارد که می‌توانید API خود را بسازید اما پیروی از اصول REST تشویق می‌شود. معماری از نمایندگی دولت انتقال است. API‌های RESTful مبتنی بر منابع هستند.

مدلهای شما نمایانگر منابع و روش‌های HTTP مانند GET، POST، PUT یا DELETE برای بازیابی، ایجاد، به روزرسانی یا حذف اشیاء هستند. کدهای پاسخ HTTP نیز در این زمینه استفاده می‌شود.

کدهای مختلف پاسخ HTTP برای نشان دادن نتیجه درخواست HTTP برگردانده می‌شوند، به عنوان مثال کدهای پاسخ XX2 برای موفقیت، XX4 برای خطاهای غیره.

متداول ترین قالب برای تبادل داده در API‌های RESTful عبارتند از JSON و XML. ما یک API REST با سریال سازی JSON برای پروژه خود ایجاد خواهیم کرد. API ما عملکردهای زیر را ارائه می‌دهد:

موضوعات را بازیابی کنید

دوره‌های موجود را بازیابی کنید

مطلوب دوره را بازیابی کنید

ثبت نام در یک دوره

ما می توانیم با ایجاد نماهای سفارشی ، یک API را از ابتدا با Django بسازیم. با این وجود چندین مژول شخص ثالث وجود دارد که ایجاد یک API را برای پروژه شما ساده می کند که محبوب ترین آنها چارچوب REST در Django است.

چارچوب نصب جنگو REST

چارچوب REST Django به شما امکان می دهد تا به راحتی API های REST را برای پروژه خود بسازید. می توانید تمام اطلاعات مربوط به چارچوب REST را در <https://www.djangoproject.org> پیدا کنید.

پوسته را باز کرده و فریم را با دستور زیر نصب کنید:

```
pip install djangorestframework
```

پرونده settings.py را ویرایش کنید و rest_framework را به تنظیمات INSTALLED_APPS برای فعال کردن برنامه اضافه کنید ، به شرح زیر:

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

سپس کد زیر را به پرونده settings.py اضافه کنید:

```
REST_FRAMEWORK = {  
    'DEFAULT_PERMISSION_CLASSES':  
        'rest_framework.permissions.DjangoModelPermissionsOrAnonReadOnly'  
}
```

با استفاده از تنظیمات REST_FRAMEWORK می توانید پیکربندی خاصی برای API خود ارائه دهید. چارچوب REST طیف گسترده ای از تنظیمات را برای پیکربندی رفتارهای پیش فرض ارائه می دهد.

تنظیم DEFAULT_PERMISSION_CLASSES مجوزهای پیش فرض برای خواندن ، ایجاد ، به روزرسانی یا حذف اشیا را مشخص می کند. ما DjangoModelPermissionOrAnonReadOnly را تنها کلاس اجازه پیش فرض می دانیم. این کلاس به سیستم مجوزهای Django متک است تا به کاربران امکان ایجاد دسترسی ، به روزرسانی یا حذف اشیاء را در حالی که دسترسی فقط به خواندن را برای کاربران ناشناس فراهم می کند. در مورد مجوزها بعداً در بخش افزودن مجوز به بازدیدها اطلاعات بیشتری کسب خواهد کرد.

برای لیست کاملی از تنظیمات موجود برای چارچوب REST ، می توانید به <https://www.djangoproject.com/en/2.0/ref/settings/#rest-framework> مراجعه کنید.

تعريف سریال ها

بعد از تنظیم چارچوب REST ، باید نحوه سریال سازی داده های ما را مشخص کنیم. داده های خروجی باید در یک قالب خاص سریالی شوند و داده های ورودی برای پردازش مجدد سریال می شوند. این چارچوب کلاسهای زیر را برای ساخت سریالایزر برای اشیاء مجزا فراهم می کند:

Serializer: سریال سازی را برای نمونه های معمولی کلاس پایتون فراهم می کند

ModelSerializer: ارائه دهنده سریال سازی برای نمونه های مدل HyperlinkedModelSerializer: همان ModelSerializer است ، اما این روابط روابطی را با پیوندها نشان می دهد تا کلیدهای اصلی ، اولین سریالایزر خود را بسازیم. ساختار پرونده زیر را در فهرست برنامه کاربردی دوره ها ایجاد کنید:

```
api/
    __init__.py
    serializers.py
```

ما تمام عملکردهای API را در داخل فهرست API ایجاد خواهیم کرد تا همه چیز به خوبی ساماندهی شود. پرونده serializer.py را ویرایش کرده و کد زیر را اضافه کنید:

```
from rest_framework import serializers
from ..models import Subject

class SubjectSerializer(serializers.ModelSerializer):

    class Meta:
        model = Subject
        fields = ['id', 'title', 'slug']
```

این سریال ساز برای مدل Subject Django's Form تعريف ModelForm است. سریال سازها با روشی مشابه کلاس‌های Meta و Django's Form می‌شوند. کلاس Meta به شما امکان می‌دهد که مدل را برای سریال کردن و زمینه هایی را برای سریال سازی مشخص کنید. در صورت تنظیم نکردن یک ویژگی فیلد ، تمام قسمتهای مدل گنجانده خواهد شد.

باید سریال ما را امتحان کنیم. خط فرمان را باز کرده و پوسته جنگو را با دستور زیر شروع کنید:

```
python manage.py shell
```

کد زیر را اجرا کنید:

```
>>> from courses.models import Subject
>>> from courses.api.serializers import SubjectSerializer
>>> subject = Subject.objects.latest('id')
>>> serializer = SubjectSerializer(subject)
>>> serializer.data
{'id': 4, 'title': 'Programming', 'slug': 'programming'}
```

در این مثال ، یک موضوع Subject دریافت می‌کنیم ، نمونه‌ای از SubjectSerializer ایجاد می‌کنیم و به داده‌های سریالی دسترسی پیدا می‌کنیم. مشاهده می‌کنید که داده‌های مدل به انواع داده بومی پایتون ترجمه شده‌اند.

درک پارس و رندرها

قبل از برگشتن در پاسخ HTTP ، داده های سریالی شده باید با فرمت خاصی ارائه شوند. به همین ترتیب ، وقتی یک درخواست HTTP دریافت می کنید ، باید داده های دریافتی را تجزیه کرده و قبل از کار با آن ، سریال سازی مجدد کنید. چارچوب REST شامل رندرها و تجزیه کننده ها برای مقابله با آن است.

باید بینیم چگونه داده های ورودی را تجزیه کنیم. کد زیر را در پوسته پایتون اجرا کنید:

```
>>> from io import BytesIO
>>> from rest_framework.parsers import JSONParser
>>> data = b'{"id":4,"title":"Programming","slug":"programming"}'
>>> JSONParser().parse(BytesIO(data))
{'id': 4, 'title': 'Programming', 'slug': 'programming'}
```

با توجه به ورودی رشته JSON ، می توانید از کلاس تجزیه گر JSON تهیه شده توسط چارچوب REST برای تبدیل آن به یک شیء پایتون استفاده کنید.

چارچوب REST همچنین شامل کلاسهای Renderer است که به شما امکان می دهد پاسخ های API را قالب بندی کنید. این چارچوب تعیین می کند که ارائه دهنده از چه طریق مذاکره محتوا استفاده کند. این گزینه هدر درخواست را بررسی می کند تا نوع محتوا مورد انتظار برای پاسخ را تعیین کند.

در صورت اختیاری ، رندر توسط پسوند قالب URL مشخص می شود. به عنوان مثال ، دسترسی به JSONRenderer باعث بازگشت پاسخ JSON می شود.

دوباره به پوسته برگردید و کد زیر را برای اجرای شیء سریال ساز از مثال سریال ساز قبلی اجرا کنید:

```
>>> from rest_framework.renderers import JSONRenderer
>>> JSONRenderer().render(serializer.data)
```

خروجی زیر را مشاهده خواهید کرد:

```
b'{"id":4,"title":"Programming","slug":"programming"}'
```

ما از JSONRenderer برای ارائه داده های سریالی شده به JSON استفاده می کنیم. به طور پیش فرض ، فریم ورک REST از دو رندر مختلف استفاده می کند: browsableAPIRenderer و JSONRenderer. حالت دوم یک رابط وب برای راحتی فراهم می کند

API خود را مرور کنید. می توانید کلاس های ارائه دهنده پیش فرض را با گزینه DEFAULT_RENDERERS_CLASSES از تنظیم REST_FRAMEWORK تغییر دهید.

می توانید اطلاعات بیشتر در مورد ارائه دهنده ها و تجزیه کنندگان را در اینجا پیدا کنید

<https://www.djangoproject.com/en/2.0/topics/rest-api/> و <https://www.djangoproject.com/en/2.0/topics/parsers/> به ترتیب.

Building list and detail views

چارچوب REST با مجموعه ای از نمایش های عمومی و ترکیبات مختلف ارائه می شود که می توانید از آنها برای ساختن نمای API استفاده کنید. اینها قابلیت بازیابی ، ایجاد ، به روزرسانی یا حذف اشیاء مدل را فراهم می کند. می توانید کلیه ترکیب ها و نمایه های عمومی ارائه شده توسط چارچوب REST را در اینجا مشاهده کنید

<https://www.djangoproject.com/en/2.0/topics/generic-views/>

باید برای بازیابی اشیاء موضوع ، یک لیست و نمایش جزئیات ایجاد کنیم. یک فایل جدید در داخل دوره ها /courses/api/views.py بنویسید. کد زیر را به آن اضافه کنید:

```
from rest_framework import generics
from ..models import Subject
from .serializers import SubjectSerializer

class SubjectListView(generics.ListAPIView):
    queryset = Subject.objects.all()
    serializer_class = SubjectSerializer

class SubjectDetailView(generics.RetrieveAPIView):
    queryset = Subject.objects.all()
    serializer_class = SubjectSerializer
```

در این کد ، ما از نمایه‌های عمومی ListAPIView و RetrieveAPIView از چارچوب REST استفاده می‌کنیم. ما یک پارامتر pk را برای نمایش جزئیات برای بازیابی شیء برای کلید اصلی داده شده در نظر می‌گیریم. هر دو دیدگاه ویژگی‌های زیر را دارند:

پایه QuerySet برای بازیابی اشیاء است: queryset

کلاس برای سریال سازی اشیاء: serializer_class

بایاید الگوهای URL را برای بازدیدهای خود اضافه کنیم. یک پرونده جدید در داخل courses/api/فهرست ایجاد کنید ، آن را urls.py بنویسید و آنرا به صورت زیر بنویسید:

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

app_name = 'courses'

urlpatterns = [
    path('subjects/',
        views.SubjectListView.as_view(),
        name='subject_list'),

    path('subjects/<pk>/',
        views.SubjectDetailView.as_view(),
        name='subject_detail'),
]
```

پرونده urls.py اصلی پروژه آموزشی را ویرایش کنید و الگوهای API را به شرح زیر وارد کنید:

```
urlpatterns = [
    # ...
    path('api/', include('courses.api.urls', namespace='api')),
]
```

ما از فضای نام API برای URL های API استفاده می کنیم. اطمینان حاصل کنید که سرور شما با دستور server python management.py اجرا شده است. پوسته را باز کرده و آدرس <http://127.0.0.1:8000/api/subjects> را با استفاده از حلقه به صورت زیر بازیابی کنید:

```
curl http://127.0.0.1:8000/api/subjects/
```

پاسخ مشابه پاسخ زیر دریافت خواهد کرد:

```
[  
    {"id":1,"title":"Mathematics","slug":"mathematics"},  
    {"id":2,"title":"Music","slug":"music"},  
    {"id":3,"title":"Physics","slug":"physics"},  
    {"id":4,"title":"Programming","slug":"programming"}  
]
```

پاسخ HTTP شامل لیستی از اشیاء Subject با فرمت JSON است.

اگر سیستم عامل شما با استفاده از حلزون نصب نشده است ، می توانید آن را از <https://curl.haxx.se/dlwiz> بارگیری کنید. به جای استفاده از حلقه ، شما همچنین می توانید از هر ابزار دیگری برای ارسال درخواست های HTTP سفارشی مانند برنامه افزودنی مرورگر مانند Postman استفاده کنید که می توانید در <https://www.getpostman.com> دریافت کنید.

را در مرورگر خود باز کنید. API قابل مشاهده چارچوب REST را به شرح زیر مشاهده خواهید کرد:

Subject List

OPTIONS

GET ▾

GET /api/subjects/

HTTP 200 OK
Allow: GET, HEAD, OPTIONS
Content-Type: application/json
Vary: Accept

```
[  
  {  
    "id": 1,  
    "title": "Mathematics",  
    "slug": "mathematics"  
  },  
  {  
    "id": 2,  
    "title": "Music",  
    "slug": "music"  
  },  
  {  
    "id": 3,  
    "title": "Physics",  
    "slug": "physics"  
  },  
  {  
    "id": 4,  
    "title": "Programming",  
    "slug": "programming"  
  }  
]
```

این رابط HTML توسط ارائه دهنده BrowsableAPIRenderer ارائه شده است.

این عناوین و محتويات نتیجه را نمایش می دهد و به شما امكان می دهد درخواست ها را انجام دهید. همچنین می توانید با وارد کردن شناسه آن در URL ، به جزئيات API برای یک موضوع دسترسی پیدا کنید.

<http://127.0.0.1:8000/api/subjects/1>

در مرورگر خود شما می توانید یک موضوع Subject را با فرمت JSON ارائه کنید.

ایجاد سریالایزرهای تو در تو

ما قصد داریم سریالایزر برای مدل Course ایجاد کنیم. پرونده api / serializer.py برنامه دوره را ویرایش کنید و کد زیر را به آن اضافه کنید:

```
from ..models import Course

class CourseSerializer(serializers.ModelSerializer):
    class Meta:
        model = Course
        fields = ['id', 'subject', 'title', 'slug', 'overview',
                  'created', 'owner', 'modules']
```

باید نگاهی بیندازیم که چگونه یک سریال یک موضوع Course می شود. پوسته را باز کنید، پوسته python management.py را اجرا کنید و کد زیر را اجرا کنید:

```
>>> from rest_framework.renderers import JSONRenderer
>>> from courses.models import Course
>>> from courses.api.serializers import CourseSerializer
>>> course = Course.objects.latest('id')
>>> serializer = CourseSerializer(course)
>>> JSONRenderer().render(serializer.data)
```

با زمینه هایی که در CourseSerializer درج کرده ایم، یک شی JSON دریافت خواهد کرد. می بینید که اشیاء مربوط به مدیر ماژول ها به عنوان لیست کلیدهای اصلی به شرح زیر سری می شوند:

```
"modules": [6, 7, 9, 10]
```

ما می خواهیم اطلاعات بیشتری در مورد هر مژول درج کنیم ، بنابراین باید سریال اشیاء مژول را بزنیم و آنها را لانه کنیم. کد قبلی فایل api / serializer.py به صورت زیر ظاهر شود:

```
from rest_framework import serializers
from ..models import Module

class ModuleSerializer(serializers.ModelSerializer):
    class Meta:
        model = Module
        fields = ['order', 'title', 'description']

class CourseSerializer(serializers.ModelSerializer):
    modules = ModuleSerializer(many=True, read_only=True)

    class Meta:
        model = Course
        fields = ['id', 'subject', 'title', 'slug', 'overview',
                  'created', 'owner', 'modules']
```

ما ModuleSerializer را برای ارائه سریال سازی برای مدل Module تعریف می کنیم. سپس ما یک ویژگی مژول به CourseSerializer اضافه می کنیم تا سریالایزر TheModuleSerializer را لانه کنیم. ما many=True را تعیین می کنیم تا نشان دهیم که چندین اشیاء را سریال می کنیم. پارامتر read_only نشان می دهد که این قسمت فقط خواندنی است و برای ایجاد یا به روزرسانی اشیاء نباید در هیچ ورودی وارد شود. پوسته را باز کرده و نمونه ای از CourseSerializer را دوباره ایجاد کنید. ویژگی داده سریال را با JSONRenderer ارائه دهید. این بار ، مژول های ذکر شده با سریال سازنده ModuleSerializer تو در تو سریال می شوند ، به شرح زیر:

```
"modules": [
    {
        "order": 0,
        "title": "Introduction to overview",
        "description": "A brief overview about the Web Framework."
    },
    {
        "order": 1,
        "title": "Configuring Django",
        "description": "How to install Django."
    },
    ...
]
```

می توانید اطلاعات بیشتر در مورد سریال ها در اینجا بخوانید

<https://www.django-rest-framework.org/api-guide/serializer>

ایجاد نماهای سفارشی

چارچوب REST کلاس APIView را فراهم می کند ، که قابلیت API را در بالای کلاس View Django ایجاد می کند. کلاس APIView با استفاده از اشیاء درخواست و پاسخ سفارشی چارچوب REST و استفاده از استثنایات APIException در بازگشت پاسخ های HTTP مناسب ، از نظر View متفاوت است. همچنین دارای سیستم احراز هویت و مجوز داخلی برای مدیریت دسترسی به نماها است.

ما قصد داریم دیداری را برای کاربران ایجاد کنیم تا در دوره ها ثبت نام کنند. پرونده api / views.py برنامه دوره را ویرایش کنید و کد زیر را به آن اضافه کنید:

```

from django.shortcuts import get_object_or_404
from rest_framework.views import APIView
from rest_framework.response import Response
from ..models import Course

class CourseEnrollView(APIView):
    def post(self, request, pk, format=None):
        course = get_object_or_404(Course, pk=pk)
        course.students.add(request.user)
        return Response({'enrolled': True})

```

نمای CourseEnrollView ثبت نام کاربر در دوره ها را کنترل می کند. کد قبلی به شرح زیر است:

1. ما یک نمای سفارشی ایجاد می کنیم که زیر کلاس ها APIView است.
2. ما یک روش ارسال () برای اقدامات POST تعریف می کنیم. هیچ روش HTTP دیگری برای این نظر مجاز نخواهد بود.
- انتظار داریم یک پارامتر pk URL حاوی شناسه یک دوره باشد.
- ما دوره را با پارامتر pk داده شده بازیابی می کنیم و در صورت یافت نشدن استثنای 404.
- کاربر فعلی را به رابطه ی many-to-many دانشجویان با موضوع Course اضافه می کنیم و یک پاسخ موفق را برمی گردانیم.

پرونده api / urls.py را ویرایش کنید و الگوی URL زیر را برای مشاهده CourseEnrollView اضافه کنید:

```

path('courses/<pk>/enroll/',
      views.CourseEnrollView.as_view(),
      name='course_enroll'),

```

از لحاظ تئوریکی ، اکنون می توانیم درخواست POST را برای ثبت نام کاربر فعلی در یک دوره انجام دهیم. با این حال ، ما باید بتوانیم کاربر را شناسایی کنیم و از دسترسی کاربران غیرمجاز به این نمای جلوگیری کنیم.

بایاید ببینیم تأیید اعتبار و مجوزهای API چگونه کار می کند.

رسیدگی به احراز هویت

چارچوب REST کلاس‌های تأیید اعتبار را برای شناسایی کاربر در حال انجام درخواست فراهم می‌کند. اگر احراز هویت موفقیت آمیز باشد، چارچوب شیء کاربر معتبر را در درخواست user قرار می‌دهد. در صورت عدم احراز هویت کاربر، به جای آن نمونه‌ای از AnonymousUser Django قرار می‌گیرد.

چارچوب REST پشتیبان هویت زیر را ارائه می‌دهد:

Autorization HTTP است. کاربر و رمز عبور در سرصفحه BasicAuthentication رمزگذاری شده با Base64 توسط مشتری ارسال می‌شود. می‌توانید اطلاعات بیشتری در مورد آن در https://en.wikipedia.org/wiki/Basic_access_authentication کسب کنید.

TokenAuthentication: این احراز هویت مبتنی بر نشانه است. از مدل توکن برای ذخیره نشانه‌های کاربر استفاده می‌شود. کاربران برای تأیید اعتبار، این نشانه را در عنوان Autorization HTTP درج می‌کنند.

SessionAuthentication: این شخص برای تأیید اعتبار از جلسات جلسه Django استفاده می‌کند. این باطن برای انجام درخواست‌های معتبر API AJAX به قسمت جلوی وب سایت شما مفید است.

RemoteUserAuthentication: این اجازه می‌دهد تا اعتبار را به سرور وب خود تغییر دهید، که یک متغیر محیط REMOTE_USER را تعیین می‌کند.

شما می‌توانید با زیر طبقه بندی کلاس BaseAuthentication REST تهیه شده است و با استفاده از روش احراز authentication()، یک احراز هویت سفارشی بسازید.

با استفاده از تنظیم DEFAULT_AUTHENTICATION_CLASSES می‌توانید تأیید اعتبار را بر اساس هر نمای تنظیم کنید یا آن را در سطح جهانی تنظیم کنید.

احراز هویت فقط کاربر در حال انجام درخواست را شناسایی می‌کند. دسترسی به نماها را مجاز یا منکر نخواهد کرد. برای محدود کردن دسترسی به بازدیدها باید از مجوزها استفاده کنید.

می‌توانید اطلاعات مربوط به تأیید اعتبار را در اینجا پیدا کنید

<https://www.djangoproject.com/en/1.11/topics/auth-api/>

بیایید BasicAuthentication را به نظر خود اضافه کنیم. پرونده api / views.py برنامه دوره‌ها را ویرایش کنید و یک CourseEnrollView را به شرح زیر اضافه کنید:

```
from rest_framework.authentication import BasicAuthentication

class CourseEnrollView(APIView):
    authentication_classes = (BasicAuthentication,)
    # ...
```

کاربران با اعتبارنامه های تعیین شده در عنوان مجوز درخواست HTTP مشخص می شوند.

افزودن مجوز به بازدیدها

چارچوب REST شامل یک سیستم اجازه برای محدود کردن دسترسی به بازدیدها است. برخی از مجوزهای داخلی چارچوب REST عبارتند از:

AllowAny: دسترسی بدون محدودیت ، صرفنظر از اینکه کاربر احراز هویت کند یا خیر.

IsAuthenticated: اجازه دسترسی فقط به کاربران معتبر را می دهد.

IsAuthenticatedOrReadOnly: دسترسی کامل به کاربران معتبر. کاربران ناشناس فقط مجاز به اجرای روش‌های خواندن مانند OPTIONS یا HEAD، GET هستند.

DjangoModelPermission: مجوزهای مرتبط با django.contrib.auth queryset. نمای به یک ویژگی queryset مانند. فقط به کاربران معتبر که دارای مجوزهای مدل اختصاصی هستند ، اجازه داده می شوند.

DjangoObjectPermission: مجوزهای جنگو بر اساس هر شی.

در صورت عدم اجازه کاربر ، معمولاً یک از کدهای خطای HTTP زیر را دریافت می کنند:

HTTP 401: غیرمجاز

HTTP 403: اجازه رد شد

می توانید اطلاعات بیشتر در مورد مجوزها را در اینجا بخوانید

<https://www.djangoproject.org/api-guide/permissions/>

پرونده api / views.py به شرح زیر CourseEnrollView کلاس را به ویژگی `_enroll` و یک ویرایش کنید:

```
from rest_framework.authentication import BasicAuthentication
from rest_framework.permissions import IsAuthenticated

class CourseEnrollView(APIView):
    authentication_classes = (BasicAuthentication,)
    permission_classes = (IsAuthenticated,)
    # ...
```

ما مجوز `IsAuthenticated` را درج می کنیم. این کار باعث می شود کاربران ناشناس از دسترسی به نمای جلوگیری کنند. اکنون می توانیم درخواست POST را به روش جدید API خود انجام دهیم.

اطمینان حاصل کنید که سرور توسعه در حال اجرا است. پوسته را باز کرده و دستور زیر را اجرا کنید:

```
curl -i -X POST http://127.0.0.1:8000/api/courses/1/enroll/
```

پاسخ زیر را دریافت خواهید کرد:

```
HTTP/1.1 401 Unauthorized
...
{"detail": "Authentication credentials were not provided."}
```

همانطور که انتظار می رود، یک کد 401 HTTP دریافت می کنیم، زیرا تأیید اعتبار نمی شود. بیایید با یکی از کاربران خود از تأیید اعتبار اولیه استفاده کنیم. دستور زیر را با جایگزین کردن دانش آموز اجرا کنید: رمزعبور با اعتبار یک کاربر موجود:

```
curl -i -X POST -u student:password  
http://127.0.0.1:8000/api/courses/1/enroll/
```

پاسخ زیر را دریافت خواهید کرد:

```
HTTP/1.1 200 OK
```

```
...  
{"enrolled": true}
```

می توانید به سایت مدیریت دسترسی پیدا کرده و بررسی کنید که کاربر اکنون در این دوره ثبت نام کرده است.

ایجاد مجموعه نمایش و روتر

ViewSets به شما امکان می دهد تعامل API خود را تعریف کنید و به چارچوب REST اجازه دهید URL ها را به صورت پویا با یک شی Router بسازد. با استفاده از مجموعه های نمایش ، می توانید از تکرار منطق برای چندین نمایش جلوگیری کنید. مشاهده مجموعه ها شامل اقدامات مربوط به ایجاد معمولی ، بازیابی ، به روزرسانی ، حذف عملیات است که شامل لیست ایجاد ، بازیابی ، به روزرسانی ، partial_update و تخریب هستند.

باید یک مجموعه نمای برای مدل Course ایجاد کنیم. پرونده api / views.py را ویرایش کنید و کد زیر را به آن اضافه کنید:

```
from rest_framework import viewsets
from .serializers import CourseSerializer

class CourseViewSet(viewsets.ReadOnlyModelViewSet):
    queryset = Course.objects.all()
    serializer_class = CourseSerializer
```

ما زیر کلاس طبقه بندی ReadOnlyModelViewSet را ارائه می دهیم ، که list and retrieve را دارد و برای هر دو لیست اشیاء بازیابی می کنیم یا یک شیء واحد را بازیابی می کنیم. پرونده urls.py / api را ویرایش کنید و یک روتربرا نمایش ما به صورت زیر ایجاد کنید:

```
from django.urls import path, include
from rest_framework import routers
from . import views

router = routers.DefaultRouter()
router.register('courses', views.CourseViewSet)

urlpatterns = [
    # ...
    path('', include(router.urls)),
]
```

ما یک شیء DefaultRouter ایجاد می کنیم و مجموعه نمایش خود را با پیشوند دوره ها ثبت می کنیم. این روتروظیفه تولید خودکار URL برای مجموعه نمای ما را به عهده دارد.

اصلی خود ، همانطور که در تصویر زیر نشان داده شده است ، لیست می کند:

Api Root

OPTIONS

GET ▾

The default basic root view for DefaultRouter

GET /api/

HTTP 200 OK

Allow: GET, HEAD, OPTIONS

Content-Type: application/json

Vary: Accept

```
{  
    "courses": "http://127.0.0.1:8000/api/courses/"  
}
```

برای بازیابی لیست دوره ها می توانید به <http://127.0.0.1:8000/api/courses> دسترسی پیدا کنید.

می توانید اطلاعات بیشتری در مورد مجموعه های مشاهده در

[./https://www.djangoproject.org/api-guide/viewsets](https://www.djangoproject.org/api-guide/viewsets) می توانید اطلاعات بیشتری در مورد روتها در اینجا پیدا کنید

[./https://www.djangoproject.org/api-guide/routers](https://www.djangoproject.org/api-guide/routers)

افزودن اقدامات اضافی برای مشاهده مجموعه ها

برای مشاهده مجموعه ها می توانید اقدامات اضافی اضافه کنید. باید نمای قبلی CourseEnrollView خود را به یک مجموعه نمایش نمای سفارشی تغییر دهیم. پرونده api / views.py را ویرایش کنید و کلاس CourseViewSet را اصلاح کنید تا به شرح زیر باشد:

```

from rest_framework.decorators import detail_route

class CourseViewSet(viewsets.ReadOnlyModelViewSet):
    queryset = Course.objects.all()
    serializer_class = CourseSerializer

    @detail_route(methods=['post'],
                 authentication_classes=[BasicAuthentication],
                 permission_classes=[IsAuthenticated])
    def enroll(self, request, *args, **kwargs):
        course = self.get_object()
        course.students.add(request.user)
        return Response({'enrolled': True})

```

ما یک روش (`enroll()`) اختصاصی اضافه می کنیم که یک عملکرد اضافی برای این مجموعه نمایش است. کد قبلی به شرح زیر است:

1. ما از دکوراتور `detail_route` چارجوب استفاده می کنیم که این عملی است که باید روی یک شی واحد انجام شود.
2. دکوراتور به ما این امکان را می دهد که ویژگی های سفارشی را برای عمل اضافه کنیم. ما مشخص می کنیم که فقط روش ارسال برای این نمایش مجاز است و کلاس های تأیید اعتبار و مجوز را تنظیم می کند.
3. ما برای بازیابی شیء `Course` از `self.get_object()` استفاده می کنیم.
- 4- کاربر فعلی را به روابط بسیار زیاد به دانشجویان اضافه می کنیم و یک پاسخ موفقیت آمیز را برمی گردیم.

پرونده `api / urls.py` را ویرایش کنید و آدرس زیر را حذف کنید ، زیرا دیگر به آن احتیاج نداریم:

```
path('courses/<pk>/enroll/',
      views.CourseEnrollView.as_view(),
      name='course_enroll'),
```

سپس پرونده سیپس / views.py را ویرایش کرده و کلاس CourseEnrollView را حذف کنید.

آدرس اینترنی برای ثبت نام در دوره ها اکنون به طور خودکار توسط روتر تولید می شود. URL همچنان یکسان است ، زیرا با استفاده از ثبت نام عملکرد ما به صورت پویا ساخته شده است.

ایجاد مجوزهای سفارشی

ما می خواهیم دانشجویان بتوانند به محتوای دوره هایی که ثبت نام کرده اند دسترسی پیدا کنند. فقط دانشجویان ثبت نام شده در یک دوره باید بتوانند به مطالب آن دسترسی پیدا کنند. بهترین راه برای انجام این کار با یک کلاس اجازه سفارشی است. Django کلاس BasePermission را فراهم می کند که به شما امکان می دهد روش های زیر را تعریف کنید:

(): بررسی مجوز سطح مشاهده has_permission

(): بررسی اجازه در سطح نمونه has_object_permission

این روشها باید صحیح برگردند تا در غیر این صورت دسترسی یا False دریافت کنند.

یک فایل جدید در داخل api/courses/permissions.py فهرست ایجاد کنید و نام آن را permissions.py بنویسید. کد زیر را به آن اضافه کنید:

```
from rest_framework.permissions import BasePermission

class IsEnrolled(BasePermission):
    def has_object_permission(self, request, view, obj):
        return obj.students.filter(id=request.user.id).exists()
```

ما کلاس BasePermission را طبقه بندی می کنیم. ما بررسی می کنیم که کاربر در حال انجام درخواست در روابط دانش آموزان با موضوع Course حضور دارد. ما قصد داریم از مجوز IsEnrroll بعدی استفاده کنیم.

سربال کردن مطالب دوره

ما باید محتوای دوره را سربال سازی کنیم. مدل محتوا شامل یک کلید خارجی عمومی است که به ما امکان می دهد اشیاء مدل های مختلف محتوا را با هم مرتبط کنیم. با این حال ، ما یک روش () render مشترک برای همه مدلها محتوا در فصل قبل اضافه کرده ایم. ما می توانیم از این روش برای ارائه محتوای ارائه شده به API استفاده کنیم.

پرونده api / serializer.py برنامه دوره را ویرایش کنید و کد زیر را به آن اضافه کنید:

```
from ..models import Content

class ItemRelatedField(serializers.RelatedField):
    def to_representation(self, value):
        return value.render()

class ContentSerializer(serializers.ModelSerializer):
    item = ItemRelatedField(read_only=True)

    class Meta:
        model = Content
        fields = ['order', 'item']
```

در این کد ، ما با طبقه بندی فیلد مربوط به سربالایزر مربوطه ارائه شده توسط چارچوب REST و غلبه بر روش Content to_representation ، یک زمینه دلخواه را تعریف می کنیم. سربالایزر ContentSerializer را برای مدل تعريف می کنیم و از زمینه دلخواه برای کلید خارجی آیتم استفاده می کنیم.

برای سربال مازول به یک سربالایزر متناوب نیاز داریم که شامل محتویات آن و یک سربالایزر دوره پیشرفته نیز هست. پرونده api / serializer.py را ویرایش کنید و کد زیر را به آن اضافه کنید:

```
class ModuleWithContentsSerializer(serializers.ModelSerializer):

    contents = ContentSerializer(many=True)

    class Meta:
        model = Module
        fields = ['order', 'title', 'description', 'contents']

class CourseWithContentsSerializer(serializers.ModelSerializer):
    modules = ModuleWithContentsSerializer(many=True)

    class Meta:
        model = Course
        fields = ['id', 'subject', 'title', 'slug',
                  'overview', 'created', 'owner', 'modules']
```

بگذارید دیدگاهی ایجاد کنیم که از عملکرد retrieve() تقلید کند، اما شامل محتوای دوره است. پرونده views.py را ویرایش کنید و روش زیر را به کلاس CourseViewSet اضافه کنید:

```
from .permissions import IsEnrolled
from .serializers import CourseWithContentsSerializer

class CourseViewSet(viewsets.ReadOnlyModelViewSet):
    # ...
    @detail_route(methods=['get'],
                  serializer_class=CourseWithContentsSerializer,
                  authentication_classes=[BasicAuthentication],
                  permission_classes=[IsAuthenticated,
                                     IsEnrolled])
    def contents(self, request, *args, **kwargs):
        return self.retrieve(request, *args, **kwargs)
```

شرح این روش به شرح زیر است:

برای مشخص کردن اینکه این عمل روی یک شی واحد انجام می شود ، از دکوراتور `detail_route` استفاده می کنیم.
ما مشخص می کنیم که فقط روش GET برای این عمل مجاز است.

ما از کلاس سریال ساز جدید `CourseWithContentsSerializer` استفاده می کنیم که شامل محتوای دوره ارائه شده است.

ما هم از مجوزهای `IsAuthenticated` و هم سفارشی `IsEnroll` استفاده می کنیم. با انجام این کار ، ما اطمینان می دهیم که فقط کاربرانی که در این دوره ثبت نام کرده اند قادر به دسترسی به مطالب آن هستند.

ما برای بازگرداندن شی `Course` از عملگر بازیابی موجود () استفاده می کنیم.

دسترسی پیدا کنید ، خواهید دید که هر مأژول این دوره شامل HTML ارائه شده برای مطالب دوره به شرح زیر است:

```
{
    "order": 0,
    "title": "Introduction to Django",
    "description": "Brief introduction to the Django Web Framework.",
    "contents": [
        {
            "order": 0,
            "item": "<p>Meet Django. Django is a high-level Python Web framework<br/>...</p>"
        },
        {
            "order": 1,
            "item": "\n<iframe width=\"480\" height=\"360\" src=\"http://www.youtube.com/embed/bgV39DlmZ2U?wmode=opaque\" frameborder=\"0\" allowfullscreen></iframe>\n"
        }
    ]
}
```

شما یک API ساده ساخته اید که به سایر سرویس ها امکان دسترسی به برنامه دوره را بصورت برنامه ای می دهد. چارچوب REST همچنین به شما امکان می دهد با ایجاد نمای ویرایش `ModelViewSet` ، ایجاد و ویرایش اشیاء را انجام دهید. ما

جنبه های اصلی چارچوب جنگو Rango را پوشش داده ایم ، اما شما می توانید اطلاعات بیشتری در مورد ویژگی های آن در مستندات گسترده آن در <https://www.djangoproject.org> کسب کنید.

خلاصه

در این فصل ، شما یک API RESTful را برای سایر سرویس ها ایجاد می کنید تا با برنامه وب خود ارتباط برقرار کنند. در فصل بعد نحوه ساختن محیط تولید با استفاده از uWSGI و NGINX به شما یاد می دهیم. همچنین یاد می گیرید که چگونه یک واسطه سفارشی را پیاده سازی کنید و دستورات مدیریت سفارشی را ایجاد کنید.

فصل سیزدهم

زندگی می کنند

در فصل قبل ، یک API RESTful برای پروژه خود ایجاد کردید.

در این فصل نحوه ساختن یک محصول را یاد خواهیم گرفت

محیطی برای پروژه ما با پوشش موضوعات زیر:

پیکربندی یک محیط تولید

ایجاد یک واسطه سفارشی

اجرای دستورات مدیریت سفارشی

ایجاد یک محیط تولید

زمان آن رسیده است که پروژه جنگو خود را در یک محیط تولید مستقر کنید. ما قصد داریم این مراحل را برای زنده ماندن پروژه خود دنبال کیم:

1. تنظیمات پروژه را برای یک محیط تولید پیکربندی کنید

2. از پک پایگاه داده PostgreSQL استفاده کنید

3. تنظیم سرور وب با uWSGI و NGINX

4- دارایی استاتیک را خدمت کنید

5- سایت خود را با SSL ایمن کنید

مدیریت تنظیمات برای چندین محیط

در پروژه های دنیای واقعی ، باید با چندین محیط سروکار داشته باشید. شما حداقل یک محیط محلی و تولید خواهید داشت اما می توانید محیط های دیگری مانند آزمایش و یا محیط های پیش تولید نیز داشته باشید. برخی از تنظیمات پروژه برای همه محیط ها متداول است ، اما برخی دیگر باید در هر محیط نادیده گرفته شوند. باید تنظیمات پروژه را برای چندین محیط تنظیم کنیم در حالی که همه چیز را به طور مرتب مرتب نگه می دارد.

در کنار فایل تنظیمات / برنامه پروژه آموزشی ، یک تنظیمات / فهرست راهنمای ایجاد کنید. فایل settings.py را به settings/base.py تغییر دهید و آن را به فهرست / تنظیمات جدید منتقل کنید. پرونده های اضافی زیر را در داخل directory ایجاد کنید تا فهرست جدید به شرح زیر باشد:

```
settings/
  __init__.py
  base.py
  local.py
  pro.py
```

این پرونده ها به شرح زیر است:

base.py: پرونده تنظیمات پایه که حاوی تنظیمات مشترک است (قبلًاً تنظیمات.py)

local.py: تنظیمات سفارشی برای محیط محلی شما

pro.py: تنظیمات سفارشی برای محیط تولید پرونده setting/base.py ویرایش کنید و خط زیر را جایگزین کنید:

```
BASE_DIR = os.path.dirname(os.path.dirname(os.path.abspath(__file__)))
```

با موارد زیر:

```
BASE_DIR =
os.path.dirname(os.path.dirname(os.path.abspath(os.path.join(__file__,
os.pardir))))
```

ما پرونده های تنظیمات خود را در یک سطح پایین تر به یک فهرست راهنمای منتقل کرده ایم ، بنابراین ما به BASE_DIR نیاز داریم تا صحیح به فهرست اصلی باشد. ما با اشاره به پوشش والدین با os.pardir به این مهم می رسیم.

پرونده setting/local.py را ویرایش کنید و خطوط زیر کد را اضافه کنید:

```
from .base import *

DEBUG = True

DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': os.path.join(BASE_DIR, 'db.sqlite3'),
    }
}
```

این پرونده تنظیمات برای محیط محلی ما است. ما تمام تنظیمات تعریف شده در پرونده base.py را وارد می کنیم و فقط تنظیمات خاصی را برای این محیط تعریف می کنیم. ما تنظیمات DEBUG و DATABASES را از پرونده base.py کپی کرده ایم زیرا این ها در هر محیط تنظیم می شوند. می توانید تنظیمات DEBUG و DATABASES را از پرونده تنظیمات base.py حذف کنید.

پرونده /setting/pro.py را ویرایش کنید و به صورت زیر ظاهر کنید:

```
from .base import *

DEBUG = False

ADMINS = (
    ('Antonio M', 'email@mydomain.com'),

)

ALLOWED_HOSTS = ['*']

DATABASES = {
    'default': {
    }
}
```

این تنظیمات مربوط به محیط تولید است. باید نگاه دقیق تری به هریک از آنها بیاندازیم:

تنظیم DEBUG به اشتباه باید برای هر محیط تولید الزامی باشد. عدم انجام این کار باعث می شود اطلاعات ردیابی و داده های پیکربندی حساس در معرض دید همه قرار بگیرند.

هنگامی که DEBUG نادرست است و یک دیدگاه یک استثنا ایجاد می کند ، تمام اطلاعات از طریق ایمیل به افرادی که در تنظیم ADMINS درج شده اند ارسال می شود. حتماً نام / آدرس الکترونیکی را با اطلاعات شخصی خود جایگزین کنید.

امنیتی است. نماد ستاره * را برای اشاره به تمام نام های میزبان درج می کنیم. نامهای میزبان مورد استفاده برای برنامه را بعداً محدود خواهیم کرد.

اطلاعات: ما فقط این تنظیمات را خالی نگه می داریم. ما قصد داریم آخر تنظیم پایگاه داده را برای تولید پوشش دهیم.

هنگام کار با چندین محیط ، یک فایل تنظیمات پایه و یک پرونده تنظیمات برای هر محیط ایجاد کنید. پرونده های تنظیمات محیط باید تنظیمات متداول را به ارت برده و از تنظیمات خاص محیط پیروی کنند.

ما تنظیمات پروژه را در یک مکان متفاوت از پرونده پیش فرض تنظیمات قرار داده ایم. شما نمی توانید با استفاده از ابزار هیچ دستوراتی را اجرا کنید مگر اینکه مازول تنظیمات مورد استفاده را مشخص کنید.

هنگام اجرای دستورات مدیریتی از پوسته ، باید یک پرچم تنظیمات اضافه کنید یا یک متغیر محیط DJANGO_SETTINGS_MODULE تنظیم کنید.

پوسته را باز کرده و دستور زیر را اجرا کنید:

```
export DJANGO_SETTINGS_MODULE=educa.settings.pro
```

این متغیر محیط DJANGO_SETTINGS_MODULE را برای جلسه پوسته فعلی تنظیم می کند. اگر می خواهید از اجرای این دستور برای هر پوسته جدید خودداری کنید ، این دستور را به پیکربندی پوسته خود در پرونده های bashrc یا bash_profile اضافه کنید. اگر این متغیر را تنظیم نکنید ، باید دستورات مدیریت ، از جمله پرچم - تنظیمات را به شرح زیر اجرا کنید:

```
python manage.py migrate --settings=educa.settings.pro
```

شما تنظیمات موفقیت آمیز را برای اداره چندین محیط ترتیب داده اید.

استفاده از PostgreSQL

در طول این کتاب ، ما بیشتر از پایگاه داده SQLite استفاده کرده ایم.

SQLite ساده و سریع است ، اما برای یک محیط تولید ، به یک پایگاه داده قدرتمندتر ، مانند MySQL ، PostgreSQL یا Oracle نیاز دارید. شما قبلًا نحوه نصب PostgreSQL و راه اندازی یک پایگاه داده PostgreSQL را در فصل 3 ، گسترش برنامه وبلاگ خود آموخته اید. در صورت نیاز به نصب PostgreSQL ، می توانید بخش نصب از Chapt er 3 ، گسترش برنامه وبلاگ خود را بخوانید.

بیایید یک کاربر PostgreSQL بسازیم. برای ایجاد کاربر پایگاه داده ، پوسته را باز کرده و دستورات زیر را اجرا کنید:

```
su postgres  
createuser -dP educa
```

از شما یک رمز عبور و مجوزهای که می خواهید به این کاربر بدهید ، از شما خواسته می شود. کلمه عبور و مجوزهای مورد نظر را وارد کرده و سپس با دستور زیر یک پایگاه داده جدید ایجاد کنید:

```
createdb -E utf8 -U educa educa
```

سپس پرونده تنظیمات / pro.py را ویرایش کرده و تنظیمات DATABASES را تغییر دهید تا به صورت زیر ظاهر شود:

```
DATABASES = {  
    'default': {  
        'ENGINE': 'django.db.backends.postgresql',  
        'NAME': 'educa',  
        'USER': 'educa',  
        'PASSWORD': '*****',  
    }  
}
```

داده های قبلی را با نام بانک اطلاعاتی و اعتبارنامه های مربوط به کاربر مورد نظر خود جایگزین کنید. بانک اطلاعاتی جدید خالی است. دستور زیر را اجرا کنید تا کلیه مهاجرت های پایگاه داده اعمال شود:

```
python manage.py migrate
```

سرانجام با دستور زیر یک superuser ایجاد کنید:

```
python manage.py createsuperuser
```

[بررسی پروژه خود](#)

جنگو شامل دستور مدیریت چک برای بررسی پروژه شما در هر زمان است. این دستور برنامه های نصب شده در پروژه Django شما را بازرسی می کند و از هرگونه خطای هشدار خارج می کند. اگر گزینه --deploy را شامل می شود، چک های اضافی که فقط برای استفاده در تولید مربوط می شوند، شروع می شوند. پس از اینکه دستور زیر را برای انجام چک انجام دهید:

```
python manage.py check --deploy
```

خروجی بدون خطای این چندین هشدار مشاهده خواهد کرد. این بدان معناست که بررسی موفقیت آمیز بوده است، اما باید هشدارها را طی کنید تا ببینید آیا کار دیگری وجود دارد که می توانید پروژه خود را برای تولید این کنید. ما قصد نداریم به این موضوع عمیق تر پردازیم، اما این را بخاطر بسپارید که باید پروژه خود را قبل از استفاده از محصول بررسی کنید تا به دنبال هر موضوع مرتبط باشد.

خدمت جنگو از طریق WSGI

سکوی استقرار اولیه جنگو WSGI است. Web Server Gateway Interface مخفف WSGI است و این استاندارد برای ارائه برنامه های Python در وب است.

هنگامی که شما با استفاده از دستور start یک پروژه جدید تولید می کنید، Django یک پرونده wsgi.py را درون فهرست پروژه خود ایجاد می کند. این پرونده حاوی یک برنامه WSGI فراخوان است، که یک نقطه دسترسی به برنامه شما است. WSGI برای اجرای پروژه شما با سرور توسعه Django و استقرار برنامه شما با سرور مورد نظر خود در محیط تولید استفاده می شود.

می توانید اطلاعات بیشتری در مورد WSGI در <https://wsgi.readthedocs.io/fa/latest> کسب کنید.

نصب uWSGI

در طول این کتاب، شما از سرور توسعه Django برای اجرای پروژه هایی در محیط محلی خود استفاده کرده اید.

با این وجود ، برای استقرار برنامه خود در یک محیط تولید ، به یک سرور وب واقعی نیاز دارید. uWSGI یک سرور برنامه بسیار سریع Python است. با برنامه Python شما با استفاده از مشخصات WSGI ارتباط برقرار می کند. Django پروژه را به فرمی ترجمه می کند که درخواست های وب را با استفاده از مشخصات WSGI ارتباط برقرار می کند. uWSGI را با استفاده از دستور زیر نصب کنید:

```
pip install uwsgi==2.0.17
```

به منظور ساخت uWSGI ، به کامپایلر C مانند GCC یا clang نیاز خواهد داشت. در یک محیط لینوکس ، می توانید آن را با دستور apt-get install build-essential نصب کنید.

اگر از macOS X استفاده می کنید ، می توانید Homebrew را با مدیر بسته brew install با استفاده از دستور uWSGI را نصب کنید. اگر می خواهید uWSGI را در ویندوز نصب کنید ، به Cygwin https://www.cygwin.com مراجعه کنید. اگر از UNIX استفاده کنید ، مطلوب است که از uWSGI در محیط های مبتنی بر Linux استفاده کنید. می توانید اسناد uWSGI را در https://uwsgi-docs.readthedocs.io / en / last / بخوانید.

پیکربندی uWSGI

می توانید uWSGI را از خط فرمان اجرا کنید. پوسته را باز کرده و دستور زیر را از فهرست پروژه آموزشی eduka اجرا کنید:

```
sudo uwsgi --module=educa.wsgi:application \
--env=DJANGO_SETTINGS_MODULE=educa.settings.prod \
--master --pidfile=/tmp/project-master.pid \
--http=127.0.0.1:8000 \
--uid=1000 \
--virtualenv=/home/env/educa/
```

اگر مجوزهای لازم را نداشته باشید ممکن است مجبور شوید sudo را به این فرمان اضافه کنید.

با استفاده از این دستور ، uWSGI را در localhost خود با گزینه های زیر اجرا می کنیم:

ما از آموزش arsa.wsgi استفاده می کنیم: برنامه WSGI قابل تماس است.

تنظیمات مربوط به محیط تولید را بارگذاری می کنیم.

ما از محیط مجازی خود استفاده می کنیم. مسیر را در گزینه virtualenv با فهرست واقعی محیط مجازی خود جایگزین کنید.
اگر از محیط مجازی استفاده نمی کنید ، می توانید از این گزینه پرش کنید.

اگر دستور را در فهرست پروژه اجرا نمی کنید ، گزینه --chdir = / path / to / arsa را با مسیر پروژه خود درج کنید.

/ http://127.0.0.1:8000 را در مرورگر خود باز کنید. شما باید HTML تولید شده را بدون بارگذاری برگه های سبک یا CSS ببینید. این امر منطقی است زیرا ما uWSGI را برای ارائه پرونده های استاتیک پیکربندی نکرده ایم.

uWSGI به شما امکان می دهد پیکربندی سفارشی را در یک پرونده ini تعریف کنید.

این راحت تر از عبور از گزینه ها از طریق خط فرمان است.

ساختار پرونده زیر را در آموزش / فهرست اصلی ایجاد کنید:

```
config/
  uwsgi.ini
```

پرونده uwsgi.ini را ویرایش کنید و کد زیر را به آن اضافه کنید:

```
[uwsgi]
# variables
projectname = educa
base = /home/projects/educa

# configuration
master = true
virtualenv = /home/env/%(projectname)
pythonpath = %(base)
chdir = %(base)
env = DJANGO_SETTINGS_MODULE=%(projectname).settings.pro
module = educa.wsgi:application
socket = /tmp/%(projectname).sock
```

در پرونده ini متغيرهای زیر را تعریف می کنیم:

نام پروژه: نام پروژه Django ما ، که دارای آموزش است.

پایه: مسیر مطلق برای پروژه آموزشی. آن را با مسیر مطلق پروژه خود جایگزین کنید.

اینها متغيرهای سفارشی هستند که در گزینه های WSGI از آنها استفاده خواهیم کرد.

شما می توانید متغيرهای دیگری را که دوست دارید تعریف کنید تا زمانی که نام متفاوت از گزینه های WSGI باشد.

گزینه های زیر را تنظیم می کنیم:

استاد: فرایند کارشناسی ارشد را فعال کنید.

virtualenv: مسیر رسیدن به محیط مجازی شما. این مسیر را با مسیری مناسب جایگزین کنید.

: مسیرهایی برای افزودن به مسیر پایتون شما . pythonpath

: مسیر دایرکتوری پروژه شما ، به طوری که uWSGI قبل از بارگذاری برنامه به آن فهرست تغییر کند. chdir

: متغیرهای محیطی. ما متغیر DJANGO_SETTINGS_MODULE را نشان می دهیم که به تنظیمات محیط تولید اشاره می کند.

ماژول WSGI برای استفاده. ما این مورد را به برنامه قابل تماس با ماژول wsgi پروژه خود تنظیم می کنیم.

سوکت: سوکت UNIX / TCP برای اتصال سرور.

گزینه سوکت برای ارتباط با برخی شخص ثالث در نظر گرفته شده است

روتر ، مانند NGINX ، در حالی که گزینه http درخواستهای HTTP ورودی را بپنیرد و آنها را به تنها مسیریابی کند. ما می خواهیم uWSGI را با استفاده از یک سوکت اجرا کنیم ، زیرا می خواهیم NGINX را به عنوان سرور وب خود پیکربندی کنیم و از طریق سوکت با uWSGI ارتباط برقرار کنیم.

می توانید لیست گزینه های موجود uWSGI را در اینجا بباید

.<https://uwsgi-docs.readthedocs.io/fa/latest/Options.html>

اکنون می توانید uWSGI را با پیکربندی سفارشی خود با استفاده از این دستور اجرا کنید:

```
uwsgi --ini config/uwsgi.ini
```

اکنون نمی توانید از مرورگر خود به نمونه uWSGI خود دسترسی پیدا کنید ، زیرا این برنامه از طریق یک سوکت اجرا می شود. بباید محیط تولید را کامل کنیم.

نصب NGINX

هنگامی که در حال ارائه یک وب سایت هستید ، باید محتوای پویا ارائه دهید ، اما باید فایلهای استاتیک مانند CSS ، پروندهای JavaScript و تصاویر را نیز سرو کنید. در حالی که uWSGI قادر به ارائه پرونده های استاتیک است ، سریار غیرضروری را به درخواست های HTTP اضافه می کند و بنابراین ، به راه اندازی یک سرور وب مانند NGINX در مقابل آن توصیه می شود.

یک سرور وب است که روی هم زمان بودن ، کارایی و استفاده کم از حافظه تمرکز دارد. NGINX همچنین به عنوان یک پروکسی معکوس عمل می کند ، و درخواست های HTTP را دریافت می کند و آنها را به سمت های مختلف منتقل می کند.

به طور کلی ، شما از یک سرور وب مانند NGINX در جلو برای ارائه کارآمد و سریع فایلهای استاتیک استفاده خواهید کرد و درخواستهای پویا را به کارگران uWSGI ارسال خواهید کرد. همچنین با استفاده از NGINX می توانید قوانینی را اعمال کرده و از قابلیت های پروکسی معکوس آن بهره مند شوید.

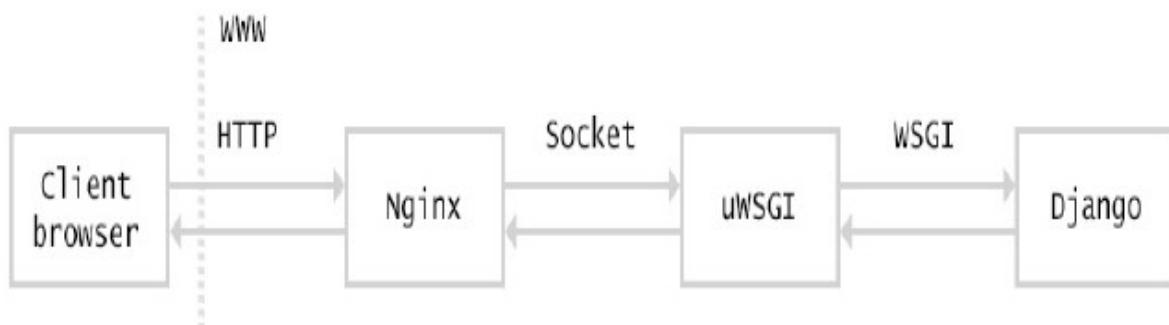
NGINX را با دستور زیر نصب کنید:

```
sudo apt-get install nginx
```

اگر از macOS X استفاده می کنید ، می توانید NGINX را با استفاده از دستور brew install nginx نصب کنید. می توانید <https://nginx.org/en/download.html> را برای ویندوز در binary NGINX پیدا کنید.

محیط تولید

نمودار زیر چگونگی نگاه محیط تولید نهایی ما را نشان می دهد:



هنگامی که مرورگر مشتری درخواست HTTP را اجرا کرد ، موارد زیر رخ می دهد:

1. NGINX درخواست HTTP را دریافت می کند.

2. در صورت درخواست پرونده استاتیک ، NGINX مستقیماً فایل استاتیک را ارائه می دهد. در صورت درخواست یک صفحه پویا ، NGINX درخواست را به uWSGI از طریق یک سوکت منتقل می کند.

3. uWSGI درخواست را برای پردازش به جنگو ارسال می کند. پاسخ HTTP حاصل به NGINX منتقل می شود ، که به نوبه خود آن را به مرورگر مشتری منتقل می کند.

پیکربندی NGINX

یک پرونده جدید را در config/ directory ایجاد کنید و نام آن را nginx.conf بگذارید.

کد زیر را به آن اضافه کنید:

```
# the upstream component nginx needs to connect to
upstream educa {
    server unix:///tmp/educa.sock;
}

server {
    listen      80;
    server_name www.educaproject.com educaproject.com;

    location / {
        include      /etc/nginx/uwsgi_params;
        uwsgi_pass  educa;
    }
}
```

این تنظیمات اصلی NGINX است. ما یک بالادست با نام آموزشی را تنظیم کرده ایم که به سوکت ایجاد شده توسط uWSGI اشاره می کند. ما از راهنمای سرور استفاده می کنیم و پیکربندی زیر را اضافه می کنیم:

به NGINX بگویید که در پورت 80 گوش کند.

نام سرور را روی هر دو سایت edukaproject.com و www.educaproject.com تنظیم کنید. NGINX درخواستهای ورودی برای هر دو دامنه را ارائه می‌دهد.

مشخص کنید که همه چیز در مسیر / مسیر باید به سوکت آموزش (uWSGI) منتقل شود. همچنین شامل پارامترهای پیش فرض پیکربندی uWSGI است که با NGINX همراه هستند.

می‌توانید اسناد NGINX را در <https://nginx.org/en/docs/> پیدا کنید.

پرونده اصلی پیکربندی NGINX در /etc/nginx/nginx.conf قرار دارد. این شامل پرونده‌های پیکربندی موجود در /etc/nginx/sites-enabled/ برای بارگذاری NGINX پرونده پیکربندی سفارشی خود، پوسته را باز کرده و پیوند نمادین را به شرح زیر ایجاد کنید:

```
sudo ln -s /home/projects/educa/config/nginx.conf /etc/nginx/sites-enabled/educa.conf
```

را با مسیر مطلق پروژه خود جایگزین کنید. اگر هنوز برنامه اجرا نشده است، یک پوسته را باز کرده و uWSGI را اجرا کنید:

```
uwsgi --ini config/uwsgi.ini
```

پوسته دوم را باز کرده و NGINX را با دستور زیر اجرا کنید:

```
service nginx start
```

از آنجاکه ما از یک نام دامنه نمونه استفاده می‌کنیم، باید آنرا به هاست محلی خود هدایت کنیم. پرونده /etc/hosts خود را ویرایش کنید و خطوط زیر را به آن اضافه کنید:

```
127.0.0.1 educaproject.com  
127.0.0.1 www.educaproject.com
```

با این کار ، ما هر دو نام میزبان را به سرور محلی خود هدایت می کنیم. در یک سرور تولید دیگر نیازی به انجام این کار نخواهید داشت ، زیرا یک آدرس IP ثابت خواهید داشت و در پیکربندی DNS دامنه خود نام میزبان خود را به سرور خود می گویید.

/ra در مرورگر خود باز کنید. شما باید بتوانید سایت خود را ببینید ، بدون اینکه هیچ دارایی استاتیکی بارگیری نشود. محیط تولید ما تقریباً آماده است.

اکنون می توانید میزبان های را که می توانند در خدمت پروژه Django شما باشند ، محدود کنید.

تنظیمات / تنظیمات فایل تنظیمات تولید را از پروژه خود ویرایش کنید و تنظیمات ALLOWED_HOSTS را به شرح زیر تغییر دهید:

```
ALLOWED_HOSTS = ['educaproject.com', 'www.educaproject.com']
```

اکنون Django فقط در صورتی که تحت هر یک از این نامهای میزبان اجرا شود ، درخواست شما را ارائه می دهد. می توانید اطلاعات بیشتر در مورد تنظیم مجاز را در <https://docs.djangoproject.com/fa/2.0/ref/settings/#allowed-hosts> بخوانید.

ارائه دارایی های استاتیک و رسانه ای

NGINX در ارائه محتوای استاتیک بسیار خوب است. برای بهترین عملکرد ، ما از NGINX برای ارائه پرونده های استاتیک در محیط تولید استفاده خواهیم کرد. ما NGINX را تنظیم خواهیم کرد تا در هر دو فایل استاتیک برنامه ما و پرونده های رسانه ای بارگذاری شده برای محتوای دوره بارگیری شود.

پرونده setting/base.py را ویرایش کنید و کد زیر را به آن اضافه کنید:

```
STATIC_ROOT = os.path.join(BASE_DIR, 'static/')
```

ما باید با جنگو دارایی استاتیک صادر کنیم. فرمان استاتیک پرونده های استاتیک را از کلیه برنامه ها کپی کرده و آنها را در فهرست STATIC_ROOT ذخیره می کند. پوسته را باز کرده و دستور زیر را اجرا کنید:

```
python manage.py collectstatic
```

این خروجی را مشاهده خواهید کرد:

```
160 static files copied to '/educa/static'.
```

اکنون پرونده config/nginx.conf را ویرایش کرده و کد زیر را در داخل بخش نامه سرور اضافه کنید:

```
location /static/ {
    alias /home/projects/educa/static/;
}
location /media/ {
    alias /home/projects/educa/media/;
}
```

به یاد داشته باشید که مسیر /home/projects/educa/ را با مسیری مطلق به فهرست پروژه خود جایگزین کنید. این بخش نامه ها به NGINX می گویند که دارایی های ایستاپی مستقر در زیر /static/ and /media/ را مستقیماً خدمت کند. این مسیرها به شرح زیر است:

/ static : این مسیر مطابق با مجموعه تنظیمات STATIC_URL است و مسیر هدف آن مطابق با مقدار تنظیمات STATIC_ROOT است. ما از آن برای ارائه پرونده های ثابت برنامه استفاده می کنیم.

/ media : این مسیر با مسیری که در تنظیمات MEDIA_URL قرار دارد مطابقت دارد و مسیر هدف آن مطابق با مقدار MEDIA_ROOT است. ما از آن برای ارائه پرونده های رسانه ای با رگذاری شده در محتوای دوره استفاده می کنیم.

برای پیگیری مسیرهای جدید ، پیکربندی NGINX را با دستور زیر بارگیری کنید:

```
service nginx reload
```

/ را در مرورگر خود باز کنید. باید بتوانید سایت خود را به درستی بارگذاری منابع استاتیک مانند صفحات و تصاویر به سبک CSS به درستی مشاهده کنید. NGINX اکنون به جای ارسال درخواست های پرونده های استاتیک به uWSGI ، مستقیماً در خدمت فایلهای استاتیک است.

عالی! شما با موفقیت NGINX را برای ارائه پرونده های استاتیک پیکربندی کرده اید.

امن سازی اتصالات با SSL

پروتکل Secure Sockets Layer (SSL) در حال تبدیل شدن به عادی برای سرویس دهی به وب سایت ها از طریق اتصال ایمن است. به شدت توصیه می شود که تحت وب سایت HTTPS به وب سایت های خود خدمت کنید. ما می خواهیم یک گواهینامه SSL را در NGINX پیکربندی کنیم تا ایمن در سایت ما باشد.

ایجاد گواهی SSL

دایرکتوری جدیدی را درون دایرکتوری پروژه آموزشی ایجاد کنید و آن را ssl بنامید. سپس با دستور زیر یک گواهی SSL از خط فرمان تولید کنید:

```
sudo openssl req -x509 -nodes -days 365 -newkey rsa:2048 -keyout  
ssl/educa.key -out ssl/educa.crt
```

ما در حال تولید یک کلید خصوصی و یک گواهی SSL بیتی 2048 بیتی هستیم که برای یک سال معتبر است. از شما خواسته می شود داده ها را به شرح زیر وارد کنید:

```
Country Name (2 letter code) [AU]:  
State or Province Name (full name) [Some-State]:  
Locality Name (eg, city) []:  
Organization Name (eg, company) [Internet Widgits Pty Ltd]:  
Organizational Unit Name (eg, section) []:  
Common Name (e.g. server FQDN or YOUR name) []: educaproject.com  
Email Address []: email@domain.com
```

می توانید داده های درخواستی را با اطلاعات شخصی خود پر کنید. مهمترین زمینه نام مشترک است. شما باید نام دامنه را برای گواهی مشخص کنید. ما از edukaproject.com استفاده می کنیم.

این کار ، در داخل `/ssl` directory ، یک فایل کلید خصوصی `eduka.key` و یک فایل آموزشی `eduka.crt` تولید می شود که گواهی واقعی است.

پیکربندی SSL برای استفاده از NGINX

پرونده `nginx.conf` را ویرایش کنید و دستورالعمل سرور را ویرایش کنید تا SSL را به شرح زیر درج کنید:

```
server {  
    listen          80;  
    listen          443 ssl;  
    ssl_certificate /home/projects/educa/ssl/educa.crt;  
    ssl_certificate_key /home/projects/educa/ssl/educa.key;  
    server_name      www.educaproject.com educaproject.com;  
    # ...  
}
```

با کد قبلی ، سرور ما هم اکنون از طریق پورت 80 و HTTPS از طریق پورت 443 به گوش می دهد. ما مسیر گواهی SSL را با `ssl_certificate` و کلید گواهی نامه با `ssl_certificate_key` نشان می دهیم.

NGINX را با دستور زیر مجدد راه اندازی کنید:

```
sudo service nginx restart
```

NGINX پیکربندی جدید را بارگیری می کند. <https://educaproject.com> را باز کنید.

با مرورگر خود باید یک پیام هشدار دهنده مشابه پیام زیر را مشاهده کنید:



This Connection is Untrusted

You have asked Firefox to connect securely to educaproject.com, but we can't confirm that your connection is secure.

Normally, when you try to connect securely, sites will present trusted identification to prove that you are going to the right place. However, this site's identity can't be verified.

What Should I Do?

If you usually connect to this site without problems, this error could mean that someone is trying to impersonate the site, and you shouldn't continue.

[Get me out of here!](#)

► Technical Details

► I Understand the Risks

If you understand what's going on, you can tell Firefox to start trusting this site's identification. Even if you trust the site, this error could mean that someone is tampering with your connection.

Don't add an exception unless you know there's a good reason why this site doesn't use trusted identification.

[Add Exception...](#)

بسته به مرورگر شما ممکن است این پیام متفاوت باشد. به شما هشدار می دهد که سایت شما از گواهی نامه معتبری استفاده نمی کند: مرورگر نمی تواند هویت سایت شما را تأیید کند. دلیل این امر این است که ما به جای گرفتن یکی از مجوزهای معتبر صدور گواهینامه (CA) ، گواهی خود را امضای کردیم. هنگامی که شما صاحب یک دامنه واقعی هستید ، می توانید برای یک CA قابل اعتماد اقدام به صدور گواهینامه SSL برای آن کنید ، تا مرورگرها بتوانند هویت آن را تأیید کنند.

اگر می خواهید برای یک دامنه واقعی یک گواهی مطمئن بدست آورید ، می توانید به پروژه Let Encrypt که توسط بنیاد لینوکس ایجاد شده است مراجعه کنید.

این یک پروژه مشترک است که با هدف ساده تر گرفتن و تجدید گواهینامه های معتبر SSL به صورت رایگان انجام می شود. می توانید اطلاعات بیشتر در <https://letsencrypt.org> پیدا کنید.

روی دکمه Add Exception کلیک کنید تا به مرورگرتان اطلاع دهید که به این گواهی اعتماد دارید. خواهید دید که مرورگر یک نماد قفل را در کنار URL به شرح زیر نشان می‌دهد:



اگر روی نماد قفل کلیک کنید، جزئیات گواهی SSL نمایش داده می‌شود.

پیکربندی پروژه برای SSL

Django دارای تنظیمات خاصی برای پشتیبانی SSL است. پرونده setting/pro.py را ویرایش کنید و تنظیمات زیر را به آن اضافه کنید:

```
SECURE_SSL_REDIRECT = True  
CSRF_COOKIE_SECURE = True
```

این تنظیمات به شرح زیر است:

SECURE_SSL_REDIRECT: آیا باید درخواست‌های HTTPS را به HTTP هدایت شوند: باید
CSRF_COOKIE_SECURE: آیا باید ایمن برای محافظت از جعل درخواست جعلی سایت تنظیم شود! شما یک محیط تولید پیکربندی کرده
برای ایجاد یک کوکی ایمن برای ارائه پروژه شما خواهد داشت.
اید که عملکرد بسیار خوبی برای ارائه پروژه شما خواهد داشت.

ایجاد یک واسطه سفارشی

شما قبل از MiddleWARE را می‌شناسید، که شامل واسطه‌هایی برای پروژه شما است. شما می‌توانید از آن به عنوان یک سیستم افزونه سطح پایین فکر کنید، به شما امکان می‌دهد قلاب‌های را که در فرایند درخواست / پاسخ اجرا می‌شوند اجرا کنید. هر واسطه مسئول برخی اقدامات خاص است که برای کلیه درخواستها یا پاسخهای HTTP اجرا خواهد شد.
از افزودن پردازش گران به وسائل مبتنی خودداری کنید، زیرا آنها در هر درخواستی اجرا می‌شوند.

هنگامی که درخواست HTTP دریافت شد ، وسط کارها به منظور ظاهر شدن در تنظیمات MiddleWARE اجرا می شوند.
هنگامی که یک پاسخ HTTP توسط Django ایجاد شده است ، پاسخ از طریق همه معیارهای میانی به ترتیب معکوس عبور می کند.

واسطه را می توان به عنوان تابعی به شرح زیر نوشت:

```
def my_middleware(get_response):

    def middleware(request):
        # Code executed for each request before
        # the view (and later middleware) are called.

        response = get_response(request)

        # Code executed for each request/response after
        # the view is called.

        return response

    return middleware
```

یک کارخانه Middleware یک فراخوانی است که call_response را فراخوانی می کند و یک واسطه را برمی گرداند. واسطه میانی یک تماس پذیر است که یک درخواست را می گیرد و پاسخی را می دهد ، دقیقاً مانند یک نمایش. در صورت وجود آخرين واسطه ذكر شده ، ممکن است call_response callable واسطه بعدی در زنجیره یا نمای واقعی باشد.

اگر هر برنامه واسطه ای بدون پاسخ دادن به get_response پاسخ خود را بازگرداند ، این روند را کوتاه می کند ، هیچ واسطه ای دیگر اجرا نمی شود (همچنین مشاهده نمی شود) و پاسخ از همان لایه هایی که درخواست از طریق آن عبور می کند باز می گردد.

ترتیب واسطه های متوسط در تنظیمات MiddleWARE بسیار مهم است زیرا یک واسطه می تواند به داده های درخواستی توسط سایر کارمندان میانی که قبلاً انجام شده اند بستگی دارد.

می توانید اطلاعات بیشتر در مورد middleware را در اینجا پیدا کنید

[./https://docs.djangoproject.com/fa/2.0/topics/http/middleware](https://docs.djangoproject.com/fa/2.0/topics/http/middleware)

ایجاد واسطه زیر دامنه

ما قصد داریم میان افزارهای سفارشی ایجاد کنیم تا دوره ها از طریق یک زیر دامنه سفارشی قابل دسترسی باشند. هر URL جزئیات جزئیات دوره ، که به نظر می رسد مانند <https://educaproject.com/course/django> ، همچنین از طریق زیر دامنه ای که از اسلامی دوره استفاده می کند ، مانند <https://django.educaproject.com> در دسترس خواهد بود. کاربران می توانند از subdomain به عنوان میانبر برای دسترسی به جزئیات دوره استفاده کنند. هر گونه درخواست به زیر دامنه ها به هر URL مربوط به جزئیات دوره مربوط هدایت می شوند.

Middlewares می تواند در هر نقطه از پروژه شما اقامت داشته باشد. با این حال ، توصیه می شود که یک فایل middleware.py را در فهرست برنامه خود ایجاد کنید.

یک پرونده جدید را درون فهرست برنامه های کاربردی دوره ایجاد کنید و آن را `middleware.py` بنویسید. کد زیر را به آن اضافه کنید:

```

from django.urls import reverse
from django.shortcuts import get_object_or_404, redirect
from .models import Course

def subdomain_course_middleware(get_response):
    """
    Provides subdomains for courses
    """
    def middleware(request):
        host_parts = request.get_host().split('.')
        if len(host_parts) > 2 and host_parts[0] != 'www':
            # get course for the given subdomain
            course = get_object_or_404(Course, slug=host_parts[0])
            course_url = reverse('course_detail',
                                 args=[course.slug])
            # redirect current request to the course_detail view
            url = '{}://{}{}'.format(request.scheme,
                                     '.'.join(host_parts[1:]),
                                     course_url)
            return redirect(url)

        response = get_response(request)
        return response

    return middleware

```

هنگامی که درخواست HTTP دریافت شد ، ما کارهای زیر را انجام می دهیم:

1. ما از میزبان مورد نظر استفاده می کنیم و آن را به قسمت هایی تقسیم می کنیم. به عنوان مثال ، اگر کاربر به mycourse.eduaproject دسترسی دارد ، ما لیستی را ["com" ، "edukaproject" ، "mycourse"] ایجاد می کنیم.
2. با بررسی اینکه آیا این شکاف بیش از دو عنصر ایجاد کرده است ، بررسی می کنیم که آیا نام میزبان شامل یک زیر دامنه است. اگر نام میزبان شامل یک زیر دامنه باشد ، و این WWW نیست ، سعی می کنیم دوره را با استفاده از مثل حلزونی که در زیر دامنه موجود است ، بگیریم.

3. اگر دوره ای پافت نشد ، ما پک استثناء HTTP 404 را مطرح می کنیم.

در غیر این صورت ، مرورگر را به URL جزئیات دوره هدایت می کنیم.

پرونده 'kurse.middleware.SubdomainCourseMiddleware' را در پایین settings.py پروژه را ویرایش کرده و 'courses.middleware.subdomain_course_middleware' را در پایین MiddleWARE به شرح زیر اضافه کنید:

```
MIDDLEWARE = [
    # ...
    'courses.middleware.subdomain_course_middleware',
]
```

اکنون میان افزار ما در هر درخواست اجرا می شود.

به یاد داشته باشید که نام های میزبان مجاز به ارائه خدمات ALLOWED_HOSTS ما در تنظیمات Django مشخص شده اند. باید این تنظیمات را تغییر دهیم به گونه ای که هر زیر دامنه احتمالی edukaproject.com مجاز به ارائه برنامه ما باشد.

پرونده تنظیمات / pro.py را ویرایش کنید و تنظیمات ALLOWED_HOSTS را به عنوان زوشه تغییر دهید:

```
ALLOWED_HOSTS = ['.edukaproject.com']
```

در خدمت چندین زیر دامنه با NGINX

ما به NGINX نیاز داریم تا بتوانیم با هر زیر دامنه ممکن به سایت خود خدمت کنیم. پرونده پیکربندی / nginx.conf را ویرایش کنید و این خط را جایگزین کنید:

```
server_name www.edukaproject.com edukaproject.com;
```

با موارد زیر:

```
server_name *.edukaproject.com edukaproject.com;
```

با استفاده از ستاره ، این قانون در مورد کلیه زیر دامنه های آموزشی apaproject.com اعمال می شود. برای آزمایش میان افزارهای محلی خود ، باید زیر دامنه های را که می خواهیم آزمایش کنیم به / etc / host ها اضافه کنیم. برای آزمایش میان افزار با یک موضوع Course ، خط زیر را به پرونده / etc / host ها اضافه کنید:

```
127.0.0.1 django.educaproject.com
```

سپس https://django.educaproject.com را در مرورگر خود باز کنید. واسطه می تواند دوره را با زیر دامنه پیدا کند و https://educaproject.com/course/django مرورگر شما را به

اجرای دستورات مدیریت سفارشی

Django به برنامه های شما اجازه می دهد تا دستورات مدیریت سفارشی را برای ابزار management.py ثبت کنند. به عنوان مثال ، ما از دستورات مدیریتی برای ایجاد پیام و کامپایل پیام در بخش 9 ، گسترش فروشگاه خود برای ایجاد و تهیه فایل های ترجمه استفاده کردیم.

یک فرمان مدیریت شامل یک مازول پایتون است که شامل یک کلاس Command است که از django.core.management.base.BaseCommand یا یکی از زیر کلاس های آن به ارث می رسد. می توانید دستورات ساده ایجاد کنید یا آنها را به عنوان آرگومانهای موقعیتی و اختیاری بسازید.

جنگو برای هر برنامه فعال در تنظیمات INSTALLED_APPS به دنبال فرمان مدیریت در مدیریت / دستورات / دایرکتوری است. هر مازول یافت شده به عنوان فرمان مدیریتی به نام خود ثبت شده است.

می توانید در مورد دستورات مدیریت سفارشی اطلاعات بیشتری کسب کنید

[./https://docs.djangoproject.com/fa/2.0/howto/custom-management-commands](https://docs.djangoproject.com/fa/2.0/howto/custom-management-commands)

ما در حال ایجاد یک فرمان مدیریت سفارشی هستیم که به دانشجویان یادآوری می کند حداقل در یک دوره ثبت نام کنند. این دستور یادآوری ایمیل را برای کاربرانی ارسال می کند که بیش از یک دوره معین ثبت شده اند و هنوز در هیچ دوره ثبت نام نکرده اند.

ساختار پرونده زیر را در فهرست برنامه دانشجویان ایجاد کنید:

```
management/  
    __init__.py  
commands/
```

```
    __init__.py  
    enroll_reminder.py
```

پرونده `enroll_reminder.py` را ویرایش کنید و کد زیر را به آن اضافه کنید:

```

import datetime
from django.conf import settings
from django.core.management.base import BaseCommand
from django.core.mail import send_mass_mail
from django.contrib.auth.models import User
from django.db.models import Count

class Command(BaseCommand):
    help = 'Sends an e-mail reminder to users registered more \
            than N days that are not enrolled into any courses yet'

    def add_arguments(self, parser):
        parser.add_argument('--days', dest='days', type=int)

    def handle(self, *args, **options):
        emails = []
        subject = 'Enroll in a course'
        date_joined = datetime.date.today() - \
                      datetime.timedelta(days=options['days'])
        users = User.objects.annotate(course_count=Count('courses_joined'))
        .filter(course_count=0, date_joined__lte=date_joined)
        for user in users:
            message = 'Dear {},\n\nWe noticed that you didn\'t\
                      enroll in any courses yet. What are you waiting\
                      for?'.format(user.first_name)
            emails.append((subject,
                           message,
                           settings.DEFAULT_FROM_EMAIL,
                           [user.email]))

        send_mass_mail(emails)
        self.stdout.write('Sent {} reminders'.format(len(emails)))

```

این دستور enroll_reminder ماست. کد قبلی به شرح زیر است:

کلاس Command از BaseCommand ارث می‌برد.

ما یک ویژگی کمک می‌کنیم. این ویژگی توضیحی کوتاه در مورد فرمانی که چاپ می‌شود در صورت اجرای فرمان python management.py help enrollReminder را ارائه می‌دهد.

ما از متدهای add_argument برای اضافه کردن آرگومان به روز استفاده می‌کنیم. این آرگومان برای مشخص کردن حداقل روزهایی که کاربر باید ثبت نام کند، بدون ثبت نام در هر دوره، برای دریافت یادآوری استفاده می‌شود.

فرمان (handel) شامل دستور واقعی است. روزهایی را برای مشخص شدن تجزیه از خط فرمان دریافت می‌کنیم. ما کاربران را که بیش از روزهای مشخص شده ثبت شده اند، بازی می‌کنیم، که هنوز در هیچ دوره ثبت نام نکرده اند. ما با حاشیه نویسی از QuerySet با تعداد کل دوره‌هایی که هر کاربر در آن ثبت نام کرده است، به این هدف دست می‌یابیم.

سرانجام، ما ایمیل را با استفاده از تابع send_mass_mail ارسال می‌کنیم، که بهینه شده است تا یک پیام SMTP برای ارسال همه ایمیل‌ها باز شود، به جای اینکه یک ایمیل را برای هر ایمیل ارسال کنید.

شما اولین فرمان مدیریت خود را ایجاد کرده اید. پوسته را باز کرده و دستور خود را اجرا کنید:

```
python manage.py enrollReminder --days=20
```

اگر یک سرور SMTP محلی در حال اجرا ندارید، می‌توانید به فصل 2، تقویت و بلاگ خود با ویژگی‌های پیشرفته نگاهی بیندازید و در آنجا تنظیمات SMTP را برای اولین پروژه Django ما پیکربندی کردیم. از طرف دیگر، می‌توانید تنظیمات زیر را به پرونده تنظیمات.py اضافه کنید تا ایمیل‌های خروجی استاندارد در حین توسعه تبدیل شوند:

```
EMAIL_BACKEND = 'django.core.mail.backends.console.EmailBackend'
```

باید دستور مدیریت خود را طوری تنظیم کنیم که سرور هر روز آن را ساعت 8 صبح اجرا کند. اگر از یک سیستم مبتنی بر UNIX مانند macOS X یا Linux استفاده می‌کنید، پوسته را باز کرده و e-crontab را اجرا کنید تا خود را ویرایش کنید. خط زیر را به آن اضافه کنید:

```
0 8 * * * python /path/to/educa/manage.py enrollReminder --days=20 --  
settings=educa.settings.pro
```

اگر با cron آشنا نیستید می توانید در <http://www.unixgeeks.org/securance/newbie/unix/cron-1.html> مقدمه ای برای cron پیدا کنید.

اگر از Windows استفاده می کنید ، می توانید کارهای خود را با استفاده از Task Scheduler برنامه ریزی کنید. می توانید اطلاعات بیشتر در مورد آن را در [https://msdn.microsoft.com/en-us/library/windows/desktop/aa383614\(v=vs.85\).aspx](https://msdn.microsoft.com/en-us/library/windows/desktop/aa383614(v=vs.85).aspx) پیدا کنید.

گزینه دیگر برای اجرای دوره ای اقدامات ، ایجاد وظایف و برنامه ریزی آنها با Celery است. به یاد داشته باشید که ما برای اجرای کارهای ناهمزمان از فصل کرفس در فصل 7 ، ساختن یک فروشگاه آنلاین استفاده کردیم.

به جای ایجاد دستورات مدیریتی و برنامه ریزی آنها با cron ، می توانید کارهای ناهمزمان ایجاد کرده و آنها را با برنامه ریز ضرب و شتم Celery اجرا کنید. می توانید اطلاعات بیشتری در مورد برنامه ریزی کارهای دوره ای با کرفس در <https://celery.readthedocs.io/fa/latest/userguide/periodic-tasks.html> کسب کنید.

جنگو همچنین شامل ابزاری برای تماس با دستورات مدیریت با استفاده از پایتون است. می توانید دستورات مدیریت را از کد خود به شرح زیر اجرا کنید:

```
from django.core import management  
management.call_command('enroll_reminder', days=20)
```

تبریک می گوییم! هم اکنون می توانید دستورات مدیریت سفارشی را برای برنامه های خود ایجاد کرده و در صورت لزوم آنها را برنامه ریزی کنید.

خلاصه

در این فصل ، شما با استفاده از uWSGI و NGINX یک محیط تولید پیکربندی کرده اید. شما همچنین یک میان افزار سفارشی را پیاده سازی کرده اید و یاد گرفته اید که چگونه دستورات مدیریت سفارشی را ایجاد کنید.

شما به انتهای این کتاب رسیده اید. تبریک می گوییم! شما مهارت های لازم برای ساخت برنامه های کاربردی وب موفق با Django را یاد گرفته اید. این کتاب شما را در روند توسعه پروژه های زندگی واقعی و ادغام جنگو با سایر فناوری ها راهنمایی کرده است. اکنون شما آماده هستید تا پروژه Django خود را بسازید ، خواه یک نمونه اولیه ساده باشد یا یک برنامه وب در مقیاس بزرگ.

با ماجراجویی بعدی جنگو موفق باشید!