TUT: Photo Organizer and Sharing App Part Django by Example

Блокнот:	IT		
Создана:	07.10.2014 22:52	Изменена:	07.10.2014 23:02
Метки:	**Django		
Источник:	http://lightbird.net/dbe/photo.html		

TUT: Photo Organizer and Sharing App Part Django by Example

Адрес источника: http://lightbird.net/dbe/photo.html

Defining the Model

As with previous tutorials, we'll start by defining a model (in photo/models.py):

```
from django.db import modelsfrom django.contrib.auth.models import Userfrom django.con
trib import admin

class Album(models.Model):
    title = models.CharField(max_length=60)
    public = models.BooleanField(default=False)
    def __unicode__(self):
        return self.title

class Tag(models.Model):
    tag = models.CharField(max_length=50)
    def __unicode__(self):
        return self.tag

class Image(models.Model):
    title = models.CharField(max_length=60, blank=True, null=True)
    image = models.FileField(upload_to="images/")
    tags = models.ManyToManyField(Tag, blank=True)
    albums = models.ManyToManyField(Album, blank=True)
    created = models.DateTimeField(auto_now_add=True)
    rating = models.IntegerField(default=50)
    width = models.IntegerField(blank=True, null=True)
    height = models.IntegerField(blank=True, null=True)
    user = models.ForeignKey(User, null=True, blank=True)

    def __unicode__(self):
        return self.image.name

class AlbumAdmin(admin.ModelAdmin):
    search_fields = ["title"]
    list_display = ["title"]

class TagAdmin(admin.ModelAdmin):
    list_display = ["tag"]

class ImageAdmin(admin.ModelAdmin):
    search_fields = ["title"]
    list_display = ["__unicode__", "title", "user", "rating", "created"]
    list_filter = ["tags", "albums"]

admin.site.register(Album, AlbumAdmin)admin.site.register(Tag, TagAdmin)admin.site.reg
ister(Image, ImageAdmin)
```

… and running: *manage.py syncdb; manage.py runserver*

We also need to create a location for uploaded images and set up our *settings.py* to point to it:

```
MEDIA_ROOT = '/home/username/dbe/media/'MEDIA_URL = 'http://127.0.0.1:8000/media/'
```

Admin will need to have its CSS, images and javascript code in this location — you'll have to copy them from *django/contrib/admin/media/*. You should also create *images* dir under *media*.

At this point, you can go ahead and add a few images in the Admin so that you have something to play with.

Photo Organizer and Sharing App Part II - Django by Example

Адрес источника: http://lightbird.net/dbe/photo2.html
Main Listing

We'll start by creating a listing of all albums with a few thumbnails of images. A lot of the code will be similar to the Blog App. Our url will be */photo/*, function will be called *main()* and we'll keep *list.html* as the template name. Here's our view:

```
from django.http import HttpResponseRedirect, HttpResponsefrom django.shortcuts import
 get_object_or_404, render_to_responsefrom django.contrib.auth.decorators import login
_requiredfrom django.core.context_processors import csrffrom django.core.paginator imp
ort Paginator, InvalidPage, EmptyPagefrom django.forms import ModelFormfrom settings i
mport MEDIA_URL

from dbe.photo.models import *

def main(request):
    """Main listing."""
    albums = Album.objects.all()
    if not request.user.is_authenticated():
        albums = albums.filter(public=True)

    paginator = Paginator(albums, 10)
    try: page = int(request.GET.get("page", '1'))
    except ValueError: page = 1

    try:
        albums = paginator.page(page)
    except (InvalidPage, EmptyPage):
        albums = paginator.page(paginator.num_pages)

    for album in albums.object_list:
        album.images = album.image_set.all()[:4]

    return render_to_response("photo/list.html", dict(albums=albums, user=request.user
,
        media_url=MEDIA_URL))
```

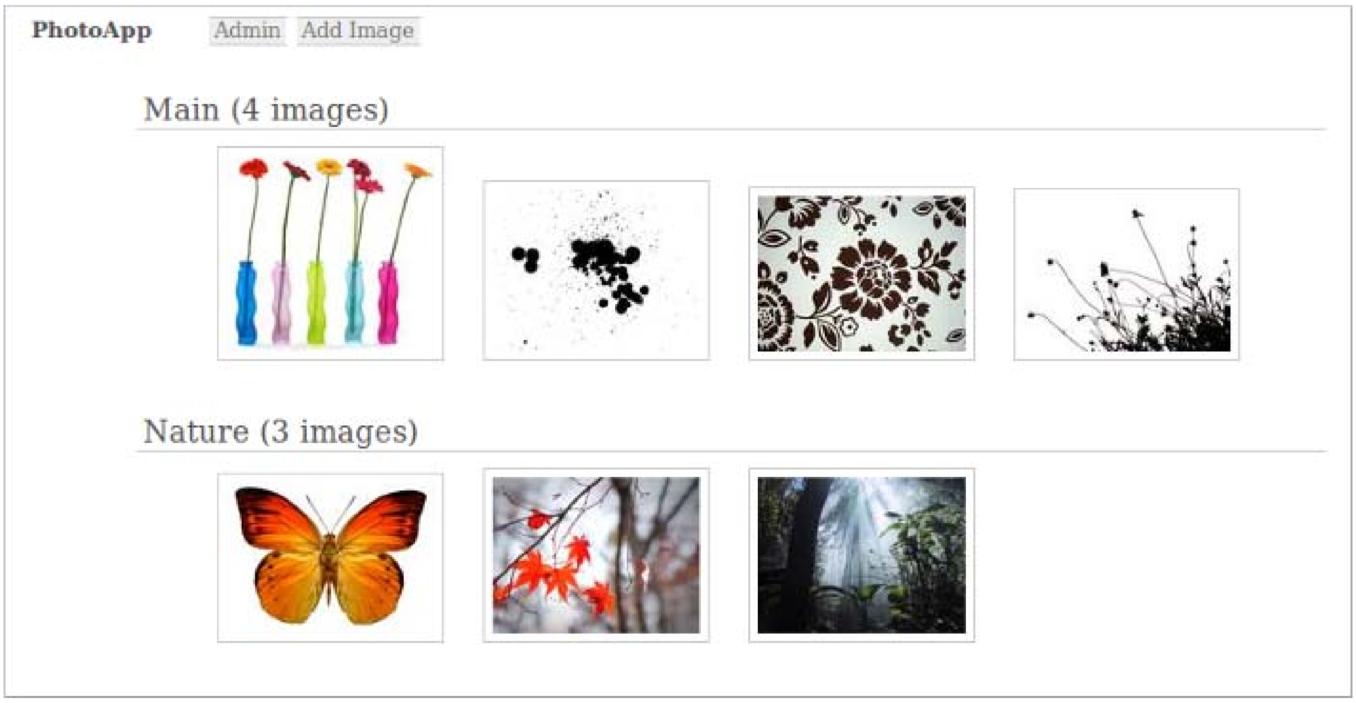
We're only listing albums that are set to *public* for users who aren't logged in. I'll speak in more detail about security at the end of the tutorial. Here is the *urlconf* line:

```
(r"", "main"),
```

...and *list.html* (don't forget to change links in *pbase.html*):

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

{% block content %}     <div class="main">

        <!-- Albums  -->        <ul>            {% for album in albums.object_list %}
                <div class="title">{{ album.title }} ({{ album.image_set.count }} imag
es)</div>                <ul>                        {% for img in album.images %}
                 <a href="{{ media_url }}{{ img.image.name }}"><img border="0" alt=""
                            src="{{ media_url }}{{ img.thumbnail2.name }}" /></a>
                {% endfor %}                     </ul>             {% endfor %}          </ul>

        <!-- Next/Prev page links  -->         {% if albums.object_list and albums.pagi
nator.num_pages > 1 %}          <div class="pagination">            <span class="step-l
inks">                  {% if albums.has_previous %}                    <a href= "?page
={{ albums.previous_page_number }}">previous &lt;&lt; </a>                 {% endif %}

                <span class="current">                      &nbsp;Page {{ albums.number
}} of {{ albums.paginator.num_pages }}                     </span>

                {% if albums.has_next %}                       <a href="?page={{ albums.
next_page_number }}"> &gt;&gt; next</a>                {% endif %}                 </span>
        </div>          {% endif %}

    </div>

{% endblock %}
```

Here's our beautiful, amazing front page (with a bit of styling added):



As you can see, we're using medium-sized thumbnails. You could also add an option to switch between the two sizes and add more sizes, as well.

Photo Organizer and Sharing App Part III - Django by Example

Адрес источника: http://lightbird.net/dbe/photo3.html

Editing Properties

Editing forms will be integrated into the album page — we'll just add a third view option. Update url and function will both be called *update*. Here are the *urlconf* lines and the changes we have to add to *album()* view:

```
(r"^(\d+)/(full|thumbnails|edit)/$", "album"),(r"^update/$", "update"),
```

```
def album(request, pk, view="thumbnails"):
    # ...
    # add list of tags as string and list of album objects to each image object
    for img in images.object_list:
        tags = [x[1] for x in img.tags.values_list()]
        img.tag_lst = join(tags, ', ')
        img.album_lst = [x[1] for x in img.albums.values_list()]

    d = dict(album=album, images=images, user=request.user, view=view, albums=Album.ob
jects.all(),
        media_url=MEDIA_URL)
    d.update(csrf(request))
    return render_to_response("photo/album.html", d)
```

We have to add quite a bit of changes to *album.html*:

```
<!-- Images  --><ul>    <div class="title">{{ album.title }}</div>        <div class="
right">        View:        <a href="{% url photo.views.album album.pk 'thumbnails' %}
">thumbnails</a>        <a href="{% url photo.views.album album.pk 'full' %}">full</a>
        <a href="{% url photo.views.album album.pk 'edit' %}">edit</a>        </div>

        {% if view == "edit" %}             <form action="{% url photo.views.update %}
" method="POST">{% csrf_token %}         {% endif %}          {% for img in images.obje
ct_list %}

            <!-- FULL VIEW  -->            {% if view == "full" %}                 <a
href="{% url photo.views.image img.pk %}"><img border="0" alt=""                     sr
c="{{ media_url }}{{ img.image.name }}"                    {% if img.width > 900 %}wid
th="900"{% endif %} /></a>              {% endif %}

            <!-- EDIT VIEW  -->            {% if view == "edit" %}

                <table>                <tr><td>                <a href="{% url photo.v
iews.image img.pk %}"><img border="0" alt=""                   src="{{ media_url }}{{
 img.thumbnail2.name }}" /></a>                    </td>                   <td>
  Title: <input type="text" name="title-{{ img.pk }}" value="{{ img.title }}" /><br />
        Tags: <input type="text" name="tags-{{ img.pk }}" value="{{ img.tag_lst }}" />
<br />        Rating:        <input size="3" type="text" name="rating-{{ img.pk }}" va
lue="{{ img.rating }}" /><br />

                {% for album in albums %}                    {{ album.title }}:
              <input type="checkbox" name="album-{{ img.pk }}" value="{{ album.pk }}"
                       {% if album.title in img.album_lst %}checked{% endif %} />
           {% endfor %}                 </td></tr></table>                <br />

            {% endif %}

            <!-- THUMBNAILS VIEW  -->            {% if view == "thumbnails" %}
         <a href="{% url photo.views.image img.pk %}"><img border="0" alt=""
          src="{{ media_url }}{{ img.thumbnail2.name }}" /></a>            {% endif %}
         {% endfor %}

        {% if view == "edit" %}             <div id="update"><input type="submit" valu
e="Update"></form></div>        {% endif %}
```

We're adding primary keys to the names of each input element to differentiate them. The rest should be fairly clear. Obviously, this UI assumes there won't be *too* many albums, otherwise you might want to use the same type of input box as for tags. I would say that 15-20 albums, maybe up to 30 should not be a problem.

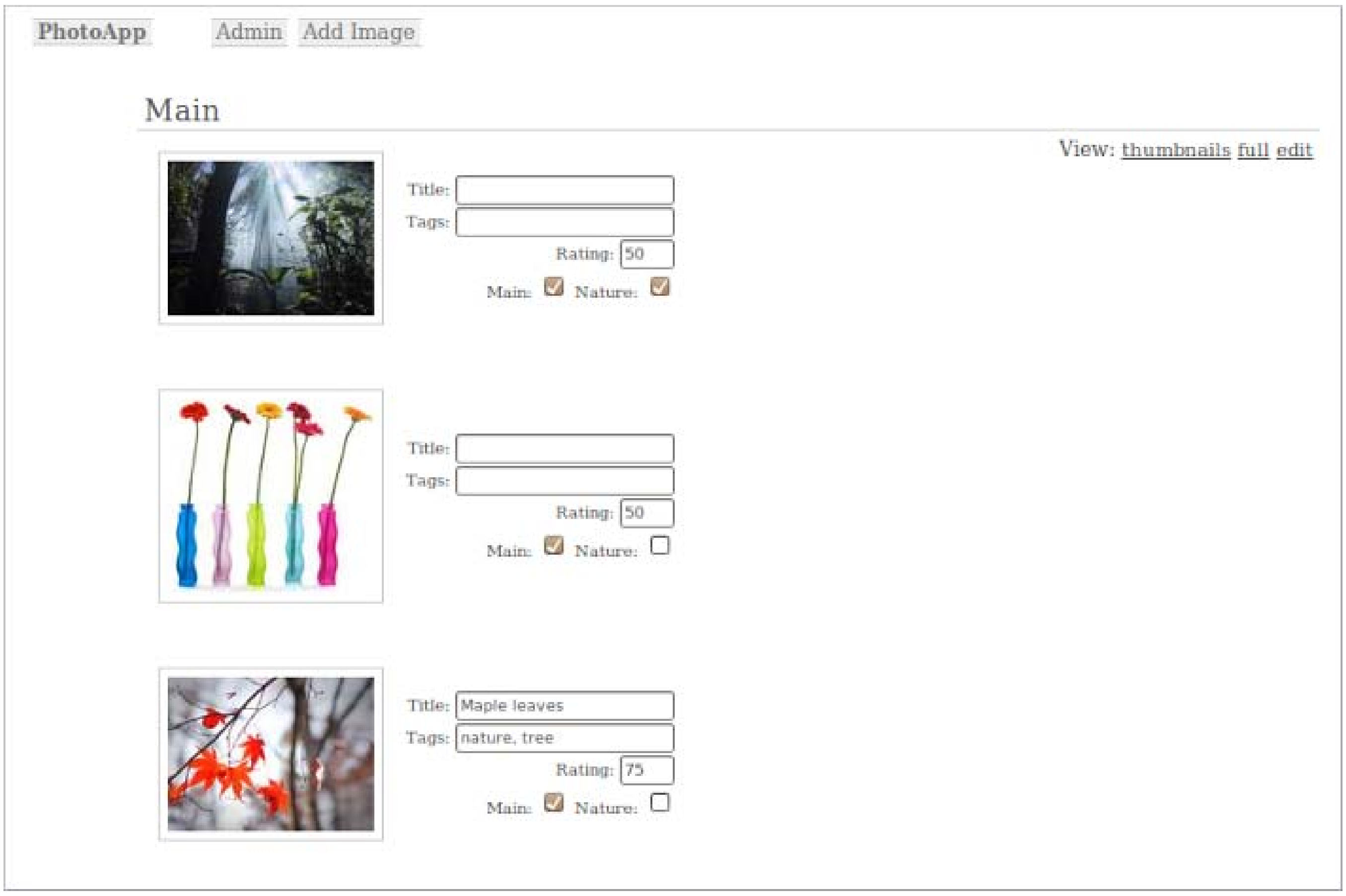
I'm sure you can't wait to see the *update()* function:

```
def update(request):
    """Update image title, rating, tags, albums."""
    p = request.POST
    images = defaultdict(dict)

    # create dictionary of properties for each image
    for k, v in p.items():
        if k.startswith("title") or k.startswith("rating") or k.startswith("tags"):
            k, pk = k.split('-')
            images[pk][k] = v
        elif k.startswith("album"):
            pk = k.split('-')[1]
            images[pk]["albums"] = p.getlist(k)

    # process properties, assign to image objects and save
    for k, d in images.items():
        image = Image.objects.get(pk=k)
        image.title = d["title"]
        image.rating = int(d["rating"])

        # tags - assign or create if a new tag!
        tags = d["tags"].split(', ')
        lst = []
        for t in tags:
            if t: lst.append(Tag.objects.get_or_create(tag=t)[0])
        image.tags = lst

        if "albums" in d:
            image.albums = d["albums"]
        image.save()

    return HttpResponseRedirect(request.META["HTTP_REFERER"], dict(media_url=MEDIA_URL
))
```

There are two interesting points I'd like to touch on here: first, take a note of how we set image.albums to the list of ids as strings — Django is smart enough to do the right thing; secondly, we're first creating a dictionary of properties for each image and then setting all of them before saving — for performance reasons, rather than setting a property at a time and saving.

It's also crucial that we create a new tag if it does not exist yet. Fortunately, Django is nice enough to provide a convenient shortcut to do just that in one line (the function returns a tuple where second value indicates if a new object was created; we're only interested in the object itself in this case).

Here's what our pretty edit interface looks like:



Main

View: thumbnails full edit

Title:
Tags:
Rating: 50
Main: ✓ Nature: ✓

Title:
Tags:
Rating: 50
Main: ✓ Nature: ☐

Title: Maple leaves
Tags: nature, tree
Rating: 75
Main: ✓ Nature: ☐

Next: part IV

Photo Organizer and Sharing App Part IV - Django by Example

Адрес источника: http://lightbird.net/dbe/photo4.html

Searching and Filtering

The one last thing we need is a page that will let us filter and sort all images by various criteria: size, title, tags, albums and ratings. We'll call it "search page" even though it will do so much more. The url, view and template will all be called *search*.

Let's start with the *urlconf* line and template:

```
(r"^search/$", "search"),
```

```
<!-- Form  --><ul>    <div class="title">Search</div>        <form action="{% url phot
o.views.search %}" method="POST">{% csrf_token %}

        <div class="form">        Title: <input type="text" name="title" value="{{ prm
.title }}" />        Filename: <input type="text" name="filename" value="{{ prm.filena
me }}" />        Tags: <input type="text" name="tags" value="{{ prm.tags }}" /><br />
       </div>

        <div class="form">        Rating:        <input size="3" type="text" name="rat
ing_from" value="{{ prm.rating_from }}" /> to        <input size="3" type="text" name=
"rating_to" value="{{ prm.rating_to }}" />        Width:        <input size="3" type="
text" name="width_from" value="{{ prm.width_from }}" /> to        <input size="3" type
="text" name="width_to" value="{{ prm.width_to }}" />        Height:        <input siz
e="3" type="text" name="height_from" value="{{ prm.height_from }}" /> to        <input
 size="3" type="text" name="height_to" value="{{ prm.height_to }}" />        </div>
```

```
        <div class="form">        {% for album in albums %}             {{ album.title
 }}:            <input type="checkbox" name="album" value="{{ album.pk }}"
    {% if album.pk in prm.album %}checked{% endif %} />        {% endfor %}

        <select name="view">            <option value="view" {% if prm.view == "view"
%}selected{% endif %}>view</option>            <option value="edit" {% if prm.view ==
"edit" %}selected{% endif %}>edit</option>        </select>

        <input type="submit" value="Apply" />        </div>

    <!-- Results  -->    <div class="title">Results</div>

        {% for img in results.object_list %}

            <!-- EDIT VIEW  -->            {% if prm.view == "edit" %}

                <table>                <tr><td>                <a href="{% url photo.v
iews.image img.pk %}"><img border="0" alt=""                src="{{ media_url }}{{
 img.thumbnail2.name }}" /></a>                    </td>                    <td>
  Title: <input type="text" name="title-{{ img.pk }}" value="{{ img.title }}" /><br />
        Tags: <input type="text" name="tags-{{ img.pk }}" value="{{ img.tag_lst }}" />
<br />        Rating:        <input size="3" type="text" name="rating-{{ img.pk }}" va
lue="{{ img.rating }}" /><br />

                {% for album in albums %}                    {{ album.title }}:
              <input type="checkbox" name="album-{{ img.pk }}" value="{{ album.pk }}"
                     {% if album.title in img.album_lst %}checked{% endif %} />
            {% endfor %}                 </td></tr></table>                <br />

            {% endif %}

            <!-- COMPACT VIEW  -->            {% if prm.view == "view" %}
    <a href="{% url photo.views.image img.pk %}"><img border="0" alt=""
     src="{{ media_url }}{{ img.thumbnail2.name }}" /></a>            {% endif %}
    {% endfor %}

        </form>

</ul>

<!-- Next/Prev page links  -->{% if results.object_list and results.paginator.num_page
s > 1 %} <div class="pagination">    <span class="step-links">        {% if results.ha
s_previous %}             <a href= "?page={{ results.previous_page_number }}">previous
 &lt;&lt; </a>        {% endif %}

        <span class="current">            &nbsp;Page {{ results.number }} of {{ result
s.paginator.num_pages }}         </span>

        {% if results.has_next %}            <a href="?page={{ results.next_page_numb
er }}"> &gt;&gt; next</a>        {% endif %}     </span></div>{% endif %}
```

...and the *search()* view:

```
@login_requireddef search(request):
    """Search, filter, sort images."""
    try: page = int(request.GET.get("page", '1'))
    except ValueError: page = 1

    p = request.POST
    images = defaultdict(dict)

    # init parameters
    parameters = {}
    keys = "title filename rating_from rating_to width_from width_to height_from heigh
```

```
t_to tags view"
    keys = keys.split()
    for k in keys:
        parameters[k] = ''
    parameters["album"] = []

    # create dictionary of properties for each image and a dict of search/filter param
eters
    for k, v in p.items():
        if k == "album":
            parameters[k] = [int(x) for x in p.getlist(k)]
        elif k in parameters:
            parameters[k] = v
        elif k.startswith("title") or k.startswith("rating") or k.startswith("tags"):
            k, pk = k.split('-')
            images[pk][k] = v
        elif k.startswith("album"):
            pk = k.split('-')[1]
            images[pk]["albums"] = p.getlist(k)

    # save or restore parameters from session
    if page != 1 and "parameters" in request.session:
        parameters = request.session["parameters"]
    else:
        request.session["parameters"] = parameters

    results = update_and_filter(images, parameters)

    # make paginator
    paginator = Paginator(results, 20)
    try:
        results = paginator.page(page)
    except (InvalidPage, EmptyPage):
        request = paginator.page(paginator.num_pages)

    # add list of tags as string and list of album names to each image object
    for img in results.object_list:
        tags = [x[1] for x in img.tags.values_list()]
        img.tag_lst = join(tags, ', ')
        img.album_lst = [x[1] for x in img.albums.values_list()]

    d = dict(results=results, user=request.user, albums=Album.objects.all(), prm=param
eters,
        media_url=MEDIA_URL)
    d.update(csrf(request))
    return render_to_response("photo/search.html", d)
```

One complication that I had to address was that the form has a large number of parameters that are submitted via *POST* request, while the paginator works through a link which is a *GET* request. One solution would be to append parameters to the link, but I think it's easier to save them in session.

The way it works is that when you submit the form, the view will save all parameters in session dictionary, filter the results and show you the first page. Once you click on the second page, parameters are loaded from session; if you re-submit the form, you'll go back to the first page again.

I split off the *update_and_filter()* function from *search()* because it was getting too big and unwieldy — I usually try to keep functions from getting longer than one screenful or so.

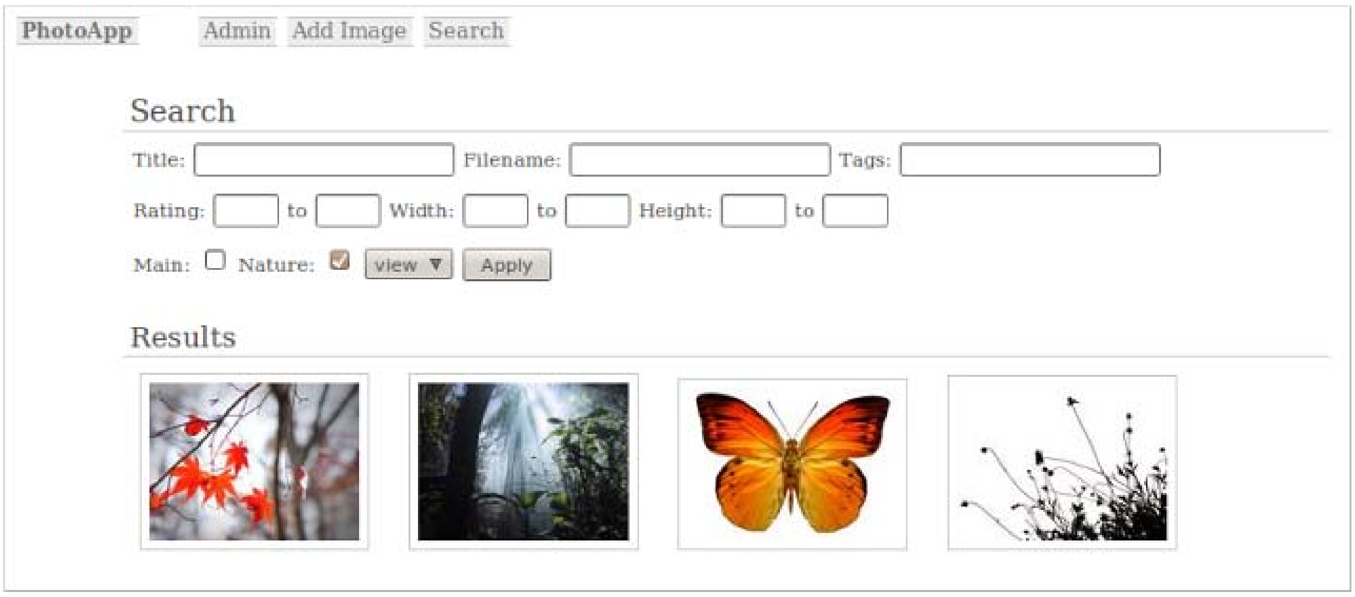
```
from django.db.models import Q

def update_and_filter(images, p):
    """Update image data if changed, filter results through parameters and return resu
lts list."""
    # process properties, assign to image objects and save
    for k, d in images.items():
        image = Image.objects.get(pk=k)
        image.title = d["title"]
        image.rating = int(d["rating"])

        # tags - assign or create if a new tag!
        tags = d["tags"].split(', ')
        lst = []
        for t in tags:
            if t: lst.append(Tag.objects.get_or_create(tag=t)[0])
        image.tags = lst

        if "albums" in d:
            image.albums = d["albums"]
        image.save()

    # filter results by parameters
    results = Image.objects.all()
    if p["title"]       : results = results.filter(title__icontains=p["title"])
    if p["filename"]    : results = results.filter(image__icontains=p["filename"])
    if p["rating_from"] : results = results.filter(rating__gte=int(p["rating_from"]))
    if p["rating_to"]   : results = results.filter(rating__lte=int(p["rating_to"]))
    if p["width_from"]  : results = results.filter(width__gte=int(p["width_from"]))
    if p["width_to"]    : results = results.filter(width__lte=int(p["width_to"]))
    if p["height_from"] : results = results.filter(height__gte=int(p["height_from"]))
    if p["height_to"]   : results = results.filter(height__lte=int(p["height_to"]))

    if p["tags"]:
        tags = p["tags"].split(', ')
        lst = []
        for t in tags:
            if t:
                results = results.filter(tags=Tag.objects.get(tag=t))

    if p["album"]:
        lst = p["album"]
        or_query = Q(albums=lst[0])
        for album in lst[1:]:
            or_query = or_query | Q(albums=album)
        results = results.filter(or_query).distinct()
    return results
```

First part of this function is the same as in *update()*; the second part has some good examples of filtering arguments: *__gte* and *__lte* filter by greater than or equal and less than or equal, respectively. Tags and Albums are filtered in a different way because it doesn't make much sense to do *AND* filtering on albums. It's a bit tricky to do *OR* filtering with unknown number of arguments — usually you could do something like this:

```
results.filter(Q(x=a) | Q(x=b) | Q(x=c))
```

In our case we don't know how many albums we'll have to deal with, therefore we have to create the *OR* query first; we also need to use the *distinct()* method to avoid duplicates.

The following screenshots illustrate various parameters in our UI:



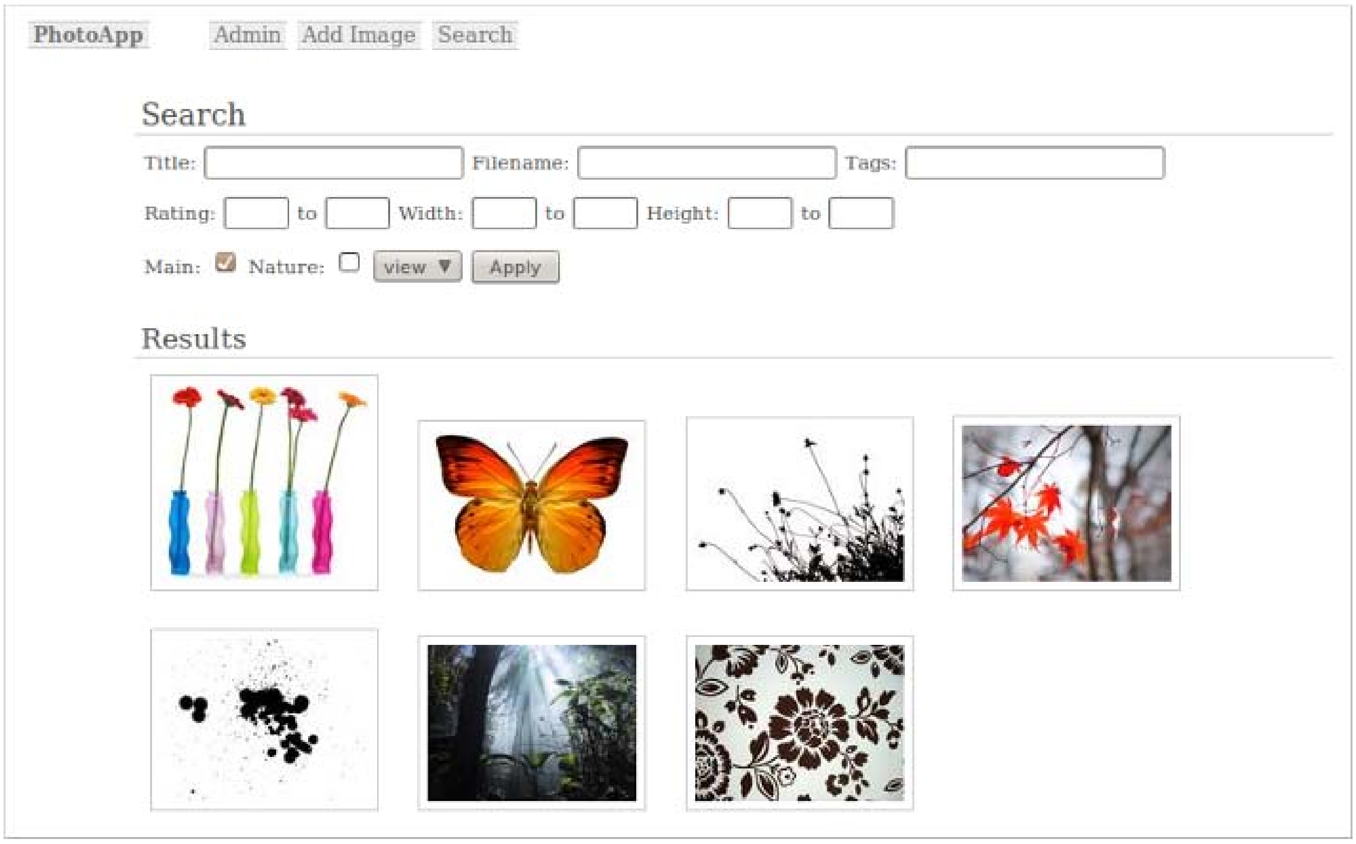
Search

Title: Filename: Tags:

Rating: to Width: to Height: to

Main: ☐ Nature: ☑

Results



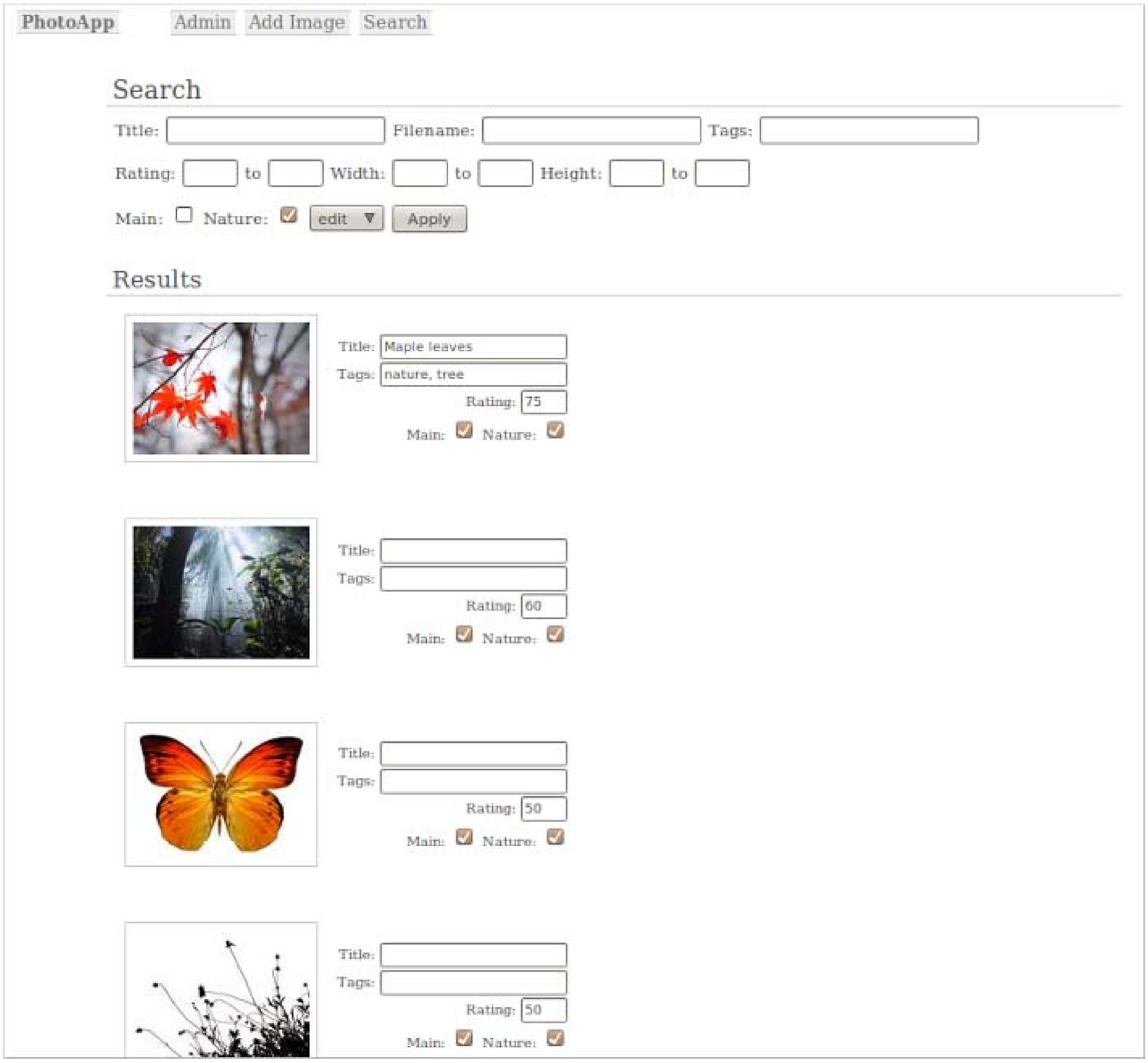
Search

Title: Filename: Tags:

Rating: to Width: to Height: to

Main: ☑ Nature: ☐

Results


PhotoApp Admin Add Image Search

Search

Title: Filename: Tags:

Rating: to Width: to Height: to

Main: ☐ Nature: ✓ edit Apply

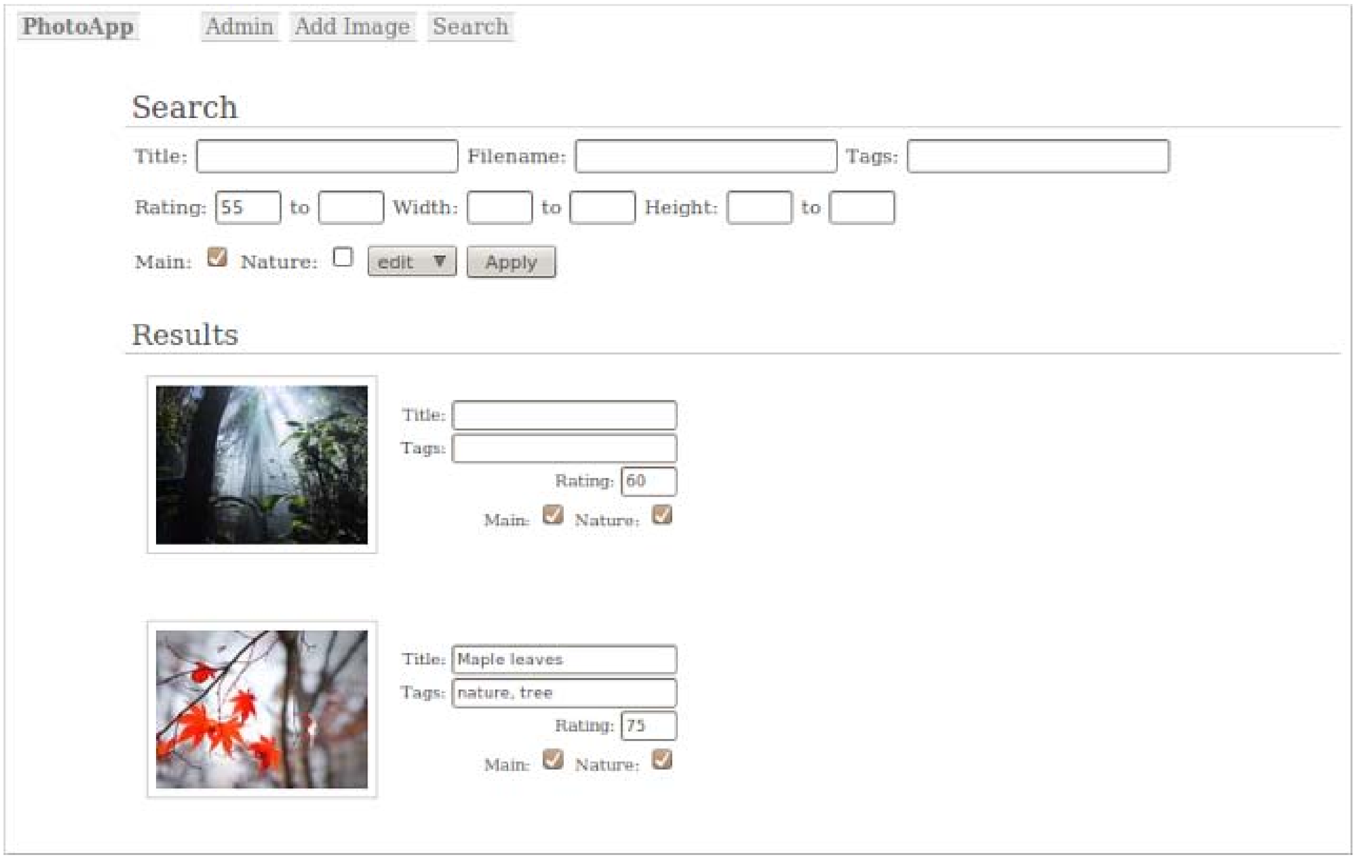
Results

Title: Maple leaves
Tags: nature, tree
Rating: 75
Main: ✓ Nature: ✓

Title:
Tags:
Rating: 60
Main: ✓ Nature: ✓

Title:
Tags:
Rating: 50
Main: ✓ Nature: ✓

Title:
Tags:
Rating: 50
Main: ✓ Nature: ✓



Search

Title: Filename: Tags:

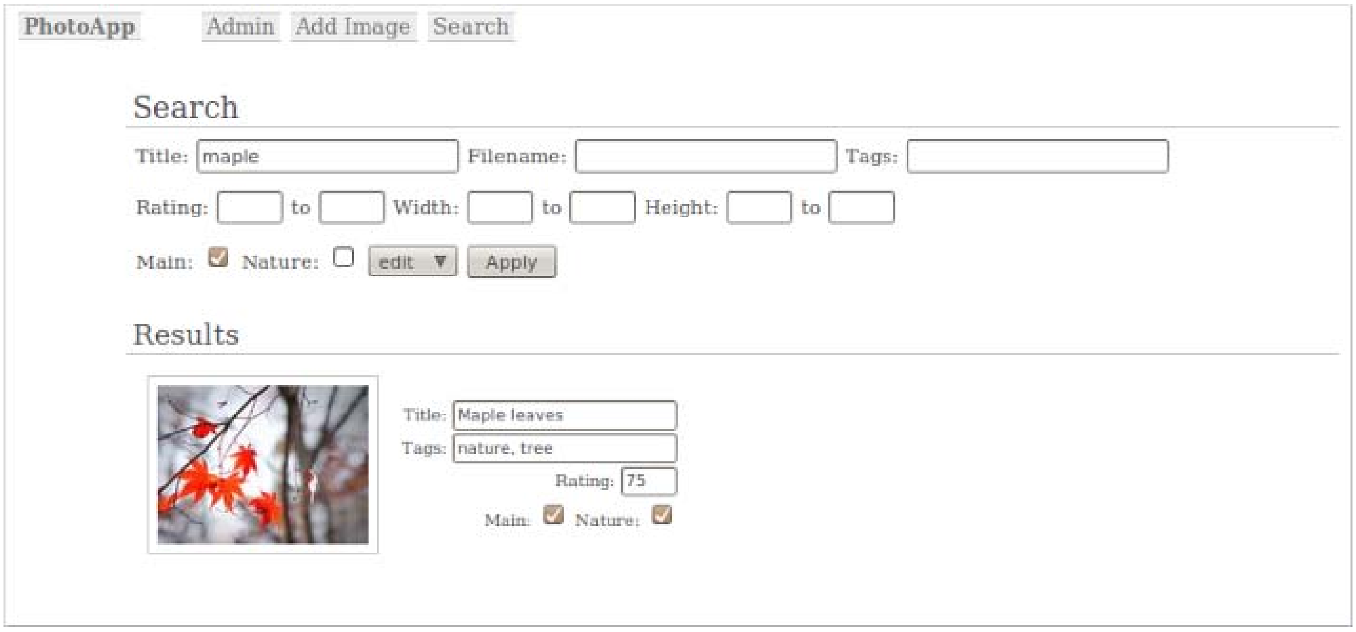
Rating: 55 to Width: to Height: to

Main: ✓ Nature: ☐ edit Apply

Results

Title:
Tags:
Rating: 60
Main: ✓ Nature: ✓

Title: Maple leaves
Tags: nature, tree
Rating: 75
Main: ✓ Nature: ✓



Search

Title: maple Filename: Tags:

Rating: to Width: to Height: to

Main: ✓ Nature: ☐ edit Apply

Results

Title: Maple leaves
Tags: nature, tree
Rating: 75
Main: ✓ Nature: ✓

Photo Organizer and Sharing App Part IV - Django by Example

Адрес источника: http://lightbird.net/dbe/photo4.html

Sorting

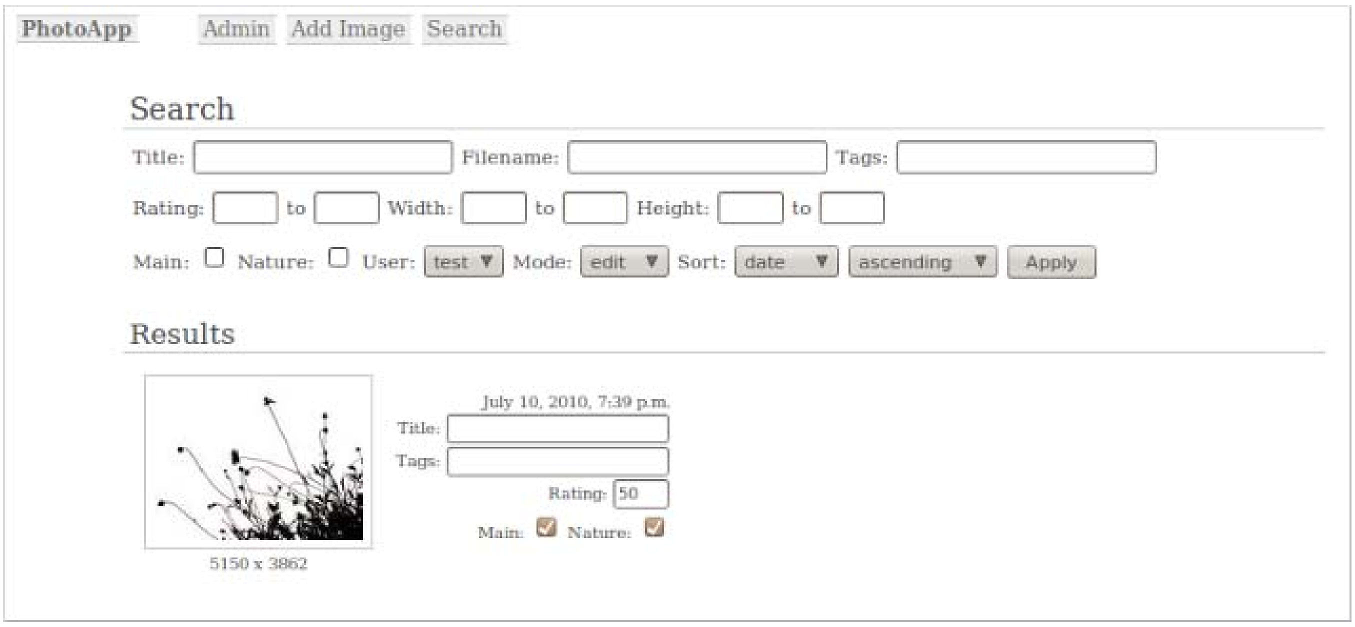
The last thing I want to add is an option to sort results by a few properties and add a *by user* filter. Everything is done in the same template and view:

```
User:<select name="user">    <option value="all" {% if prm.user == "all" %}selected{% 
endif %}>all</option>    {% for user in users %}         <option value="{{ user.pk }}"
 {% if prm.user == user.pk %}selected{% endif %}>            {{ user.username }}</opti
on>    {% endfor %} </select>

Sort:<select name="sort">    <option value="created" {% if prm.sort == "created" %}sel
ected{% endif %}>date</option>    <option value="rating" {% if prm.sort == "rating" %}
selected{% endif %}>rating</option>    <option value="width" {% if prm.sort == "width"
 %}selected{% endif %}>width</option>    <option value="height" {% if prm.sort == "hei
ght" %}selected{% endif %}>height</option></select>

<select name="asc_desc">    <option value="asc" {% if prm.sort == "asc" %}selected{% e
ndif %}>ascending</option>    <option value="desc" {% if prm.sort == "desc" %}selected
{% endif %}>descending</option></select>
```

Hopefully you can see where this code needs to be inserted; if not, link to full sources will be provided at the end of this part.

```
def search(request):
    # ...

    keys = "title filename rating_from rating_to width_from width_to height_from heigh
t_to tags view"\
        " user sort asc_desc"
    keys = keys.split()

    # ...

    for k, v in p.items():
        if k == "album":
            parameters[k] = [int(x) for x in p.getlist(k)]
        elif k == "user":
            if v != "all": v = int(v)
            parameters[k] = v

    # ...

    d = dict(results=results, user=request.user, albums=Album.objects.all(), prm=param
eters,
             users=User.objects.all(), media_url=MEDIA_URL)

def update_and_filter(images, p):

    # ...

    # sort and filter results by parameters
    order = "created"
    if p["sort"]: order = p["sort"]
    if p["asc_desc"] == "desc": order = '-' + order

    results = Image.objects.all().order_by(order)
    if p["user"] and p["user"] != "all"    : results = results.filter(user__pk=int(p["
user"]))

    # ...
```

I've also added a bit of image data to *edit* view mode:


PhotoApp Admin Add Image Search

Search

Title: Filename: Tags:

Rating: to Width: to Height: to

Main: ☐ Nature: ☐ User: test Mode: edit Sort: date ascending Apply

Results

5150 x 3862

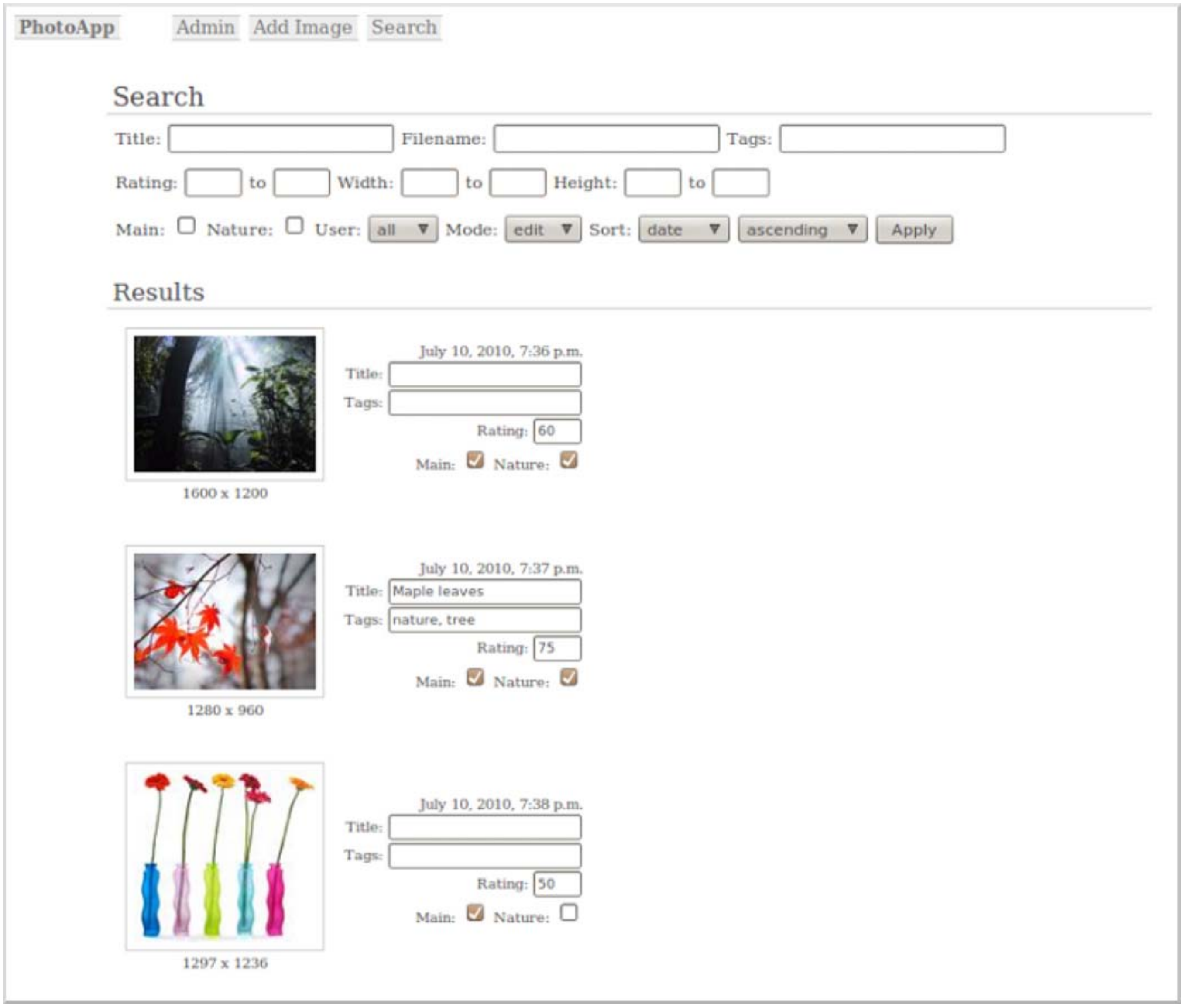
July 10, 2010, 7:39 p.m.

Title:

Tags:

Rating: 50

Main: ✓ Nature: ✓


PhotoApp Admin Add Image Search

Search

Title: Filename: Tags:

Rating: to Width: to Height: to

Main: ☐ Nature: ☐ User: all Mode: edit Sort: date ascending Apply

Results

1600 x 1200

July 10, 2010, 7:36 p.m.

Title:

Tags:

Rating: 60

Main: ✓ Nature: ✓

1280 x 960

July 10, 2010, 7:37 p.m.

Title: Maple leaves

Tags: nature, tree

Rating: 75

Main: ✓ Nature: ✓

1297 x 1236

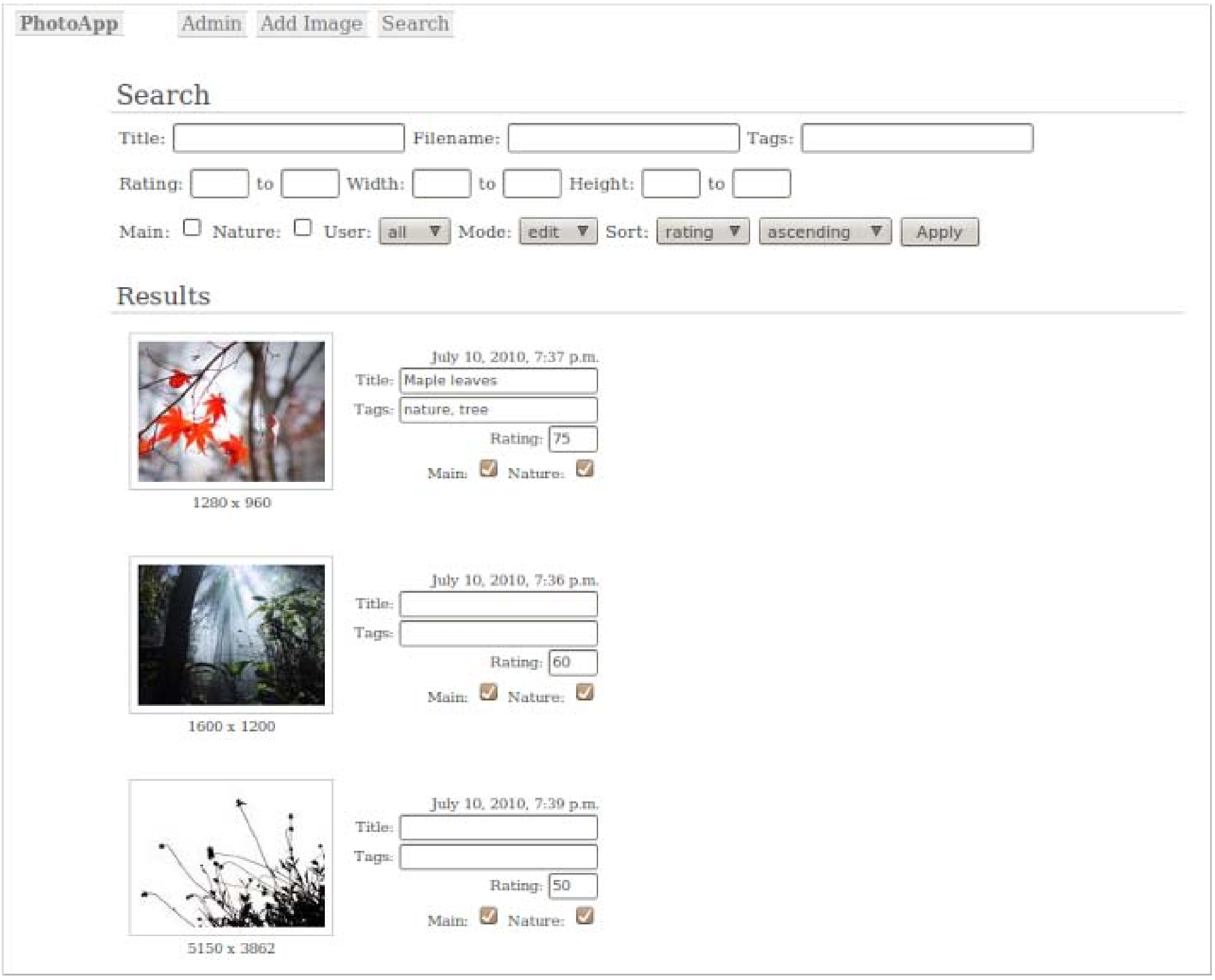
July 10, 2010, 7:38 p.m.

Title:

Tags:

Rating: 50

Main: ✓ Nature: ☐



Search

Title: Filename: Tags:

Rating: to Width: to Height: to

Main: ☐ Nature: ☐ User: all Mode: edit Sort: rating ascending

Results

July 10, 2010, 7:37 p.m.
Title: Maple leaves
Tags: nature, tree
Rating: 75
Main: ✓ Nature: ✓
1280 x 960

July 10, 2010, 7:36 p.m.
Title:
Tags:
Rating: 60
Main: ✓ Nature: ✓
1600 x 1200

July 10, 2010, 7:39 p.m.
Title:
Tags:
Rating: 50
Main: ✓ Nature: ✓
5150 x 3862

Download full tutorial sources

I've added a bit of very basic, "light-duty" security to this App. Make no mistake: a determined and technically sophisticated user will be able to to look at the images in a non-public album: all images are available as simple links under */media/images/* (although he'll have to guess the filenames since */media/* does not allow listing of directory contents).

I won't add the following code to the tutorial, but the way to avoid this would be to store images outside of */media/* and have Django serve images by itself (this is not a very efficient method but it may be acceptable for a small app). Here is a small snippet of a view that serves an image file from disk:

```
def get_image(request, fn):
    fn = fn.encode("utf-8")
    imgdir = pjoin(MEDIA_ROOT, "../images")
    ifn = pjoin(imgdir, fn)
    return HttpResponse(open(ifn).read(), mimetype='image/jpeg')
```

Images used in the tutorial were made and copyrighted by:

http://www.sxc.hu/profile/reuben4eva http://www.sxc.hu/profile/mike62 http://www.sxc.hu/profile/paaseiland
http://www.sxc.hu/profile/tijmen http://www.sxc.hu/profile/shark001 http://www.sxc.hu/profile/jamie84
http://www.sxc.hu/profile/pipp