Типичные сценарии web-приложений

Типичные сценарии

- Отображение объекта
- Отображение списка объектов
- Обработка форм и изменение объектов
- Авторизация и сессии пользователей
- Запуск фоновых процессов
- Интеграция с внешними системами

Отображение объекта

Маршрут в urls.py

```
# blog/urls.py
urlpatterns = [
    url(r'^post/(?P<slug>\w+)/$', post_details,
        name='post-details'),
    url(r'^tag/^(?P<slug>\w+)/$', tag_details,
        name='tag-details'),
]
```

Часто для на объекты ссылаются не по **id**, а по **slug**. Это позволяет строить более запоминающийся для человека URL.

Базовый view

```
from django.http import Http404
from django.shortcuts import render

def post_details(request, slug):
    try:
        post = Post.objects.get(slug=slug)
    except Post.DoesNotExist:
        raise Http404
    return render(request, 'blog/post_details.html', {
        'post':     post,
    })
```

Использование shortcut'ов

```
from django.shortcuts import render, get_object_or_404
from django.views.decorators.http import require_GET

@require_GET
def post_details(request, slug):
    post = get_object_or_404(Post, slug=slug)
    return render(request, 'blog/post_details.html', {
        'post':     post,
    })
```

Отображение связанных сущностей

```
def post_details(request, slug):
    post = get_object_or_404(Post, slug=slug)
    try:
        vote = post.votes.filter(user=request.user)[0]
    except Vote.DoesNotExist:
        vote = None
    return render(request, 'blog/post_details.html', {
        'post':     post,
        'category': post.category,
        'tags':     post.tags.all()[:],
        'vote':     vote,
    })
```

Отображение связанных сущностей

```
<h1>{{ post.category.title }} - {{ post.title }}</h1>
{% for tag in post.tags.all %}
    <a href="{{ tag.get_url }}">{{ tag }}</a>
{% endfor %}
```

Методы в моделях

```
from django.core.urlresolvers import reverse

class Tag(models.Model):
    slug  = models.SlugField(unique=True)
    title = models.CharField(max_length=64)

    def get_url(self):
        return reverse('blog:tag-details',
            kwargs={'slug': self.slug})

    def __unicode__(self):
        return self.title
```

Отображение списка объектов

Постраничное отображение

```
from django.core.paginator import Paginator
def post_list_all(request):
    posts = Post.objects.filter(is_published=True)
    limit = request.GET.get('limit', 10)
    page = request.GET.get('page', 1)
    paginator = Paginator(posts, limit)
    paginator.baseurl = '/blog/all_posts/?page='
    page = paginator.page(page)  # Page
    return render(request, 'blog/post_by_tag.html', {
        posts:     page.object_list,
        paginator: paginator, page: page,
    })
```

Шаблон paginator

```
<nav><ul class="pagination">
{% for p in paginator.page_range %}
  {% if p.number == page.number %}
  <li class="active">
  {% else %}
  <li>
  {% endif %}
    <a href="{{ paginator.baseurl }}{{ p.number }}">
    {{ p.number }}</a>
  </li>
{% endfor %}
</ul></nav>
```

django.core.paginator.Paginator

Свойства

- `count` - полное число объектов
- `num_pages` - полное число страниц
- `page_range` - список страниц, например `[1, 2, 3, 4]`

Методы

- `page(n)` - получить n-тый объект `Page`

django.core.paginator.Page

Свойства

- `object_list` - список объектов на странице
- `number` - порядковый номер страницы

Методы

- `has_nex()` / `has_previous()` - наличие соседней страницы
- `next_page_number()` / `next_page_number()`
- `start_index()` / `end_index()` - номера первого и последнего объектов на странице

Best practices

- Проверять валидность параметров `page` и `limit`
- Отображать 404 ошибку при некорректных параметрах
- Ограничивать максимальное значение `limit` <= 1000
- Обрабатывать «пустую» последнюю страницу

```
def paginate(request, qs):
    try:
        limit = int(request.GET.get('limit', 10))
    except ValueError:
        limit = 10
    if limit > 100:
        limit = 10
    try:
        page = int(request.GET.get('page', 1))
    except ValueError:
        raise Http404
    paginator = Paginator(qs, limit)
    try:
        page = paginator.page(page)
    except EmptyPage:
        page = paginator.page(paginator.num_pages)
    return page
```

Progressive loading

Progressive loading

Постраничная загрузка хорошо работает в тех случаях, когда легко определить число объектов подходящих под поисковый запрос. Другими словами - когда можно составить эффективный SQL запрос.

Иногда это сделать сложно, например: «отображать все посты в порядке добавления, но не больше одного поста из одной категории подряд».

Свой ModelManager

```
class PostManager(model.Manager):
    def main(self, since, limit=10):
        qs = self.order_by('-id')
        res = []
        if since is not None:
            qs = qs.filter('id__lt'=since)
        for p in qs:
            if len(res):
                res.append(p)
            elif res[-1].category != p.category:
                res.append(p)
            if len(res) >= limit:
                break
        return res
```

view и шаблон

```
def post_list_main(request):
    since = request.GET.get('since')
    posts = Post.objects.main(since)
    return render(request, 'blog/post_main.html', {
        posts: posts,
        since: posts[-1].id,
    })
```

```
<a href="/blog/main/?since={{ since }}">Далее</a>
```