

ListViewで一覧ページ作成

前はUpdateViewを使用して編集機能を作成したが、今回はListViewを使用して一覧ページを作成する。

これまでは関数ベースビューを使用して以下のように書いていた。

myproject/urls.py

```
path('', views.home, name='home'),
```

boards/views.py

```
from django.shortcuts import render
from .models import Board

def home(request):
    boards = Board.objects.all()
    return render(request, 'home.html', {'boards': boards})
```

上記をListViewを使用した方法で書き換えると以下ようになる。

myproject/urls.py

```
path('', views.BoardListView.as_view(), name='home'),
```

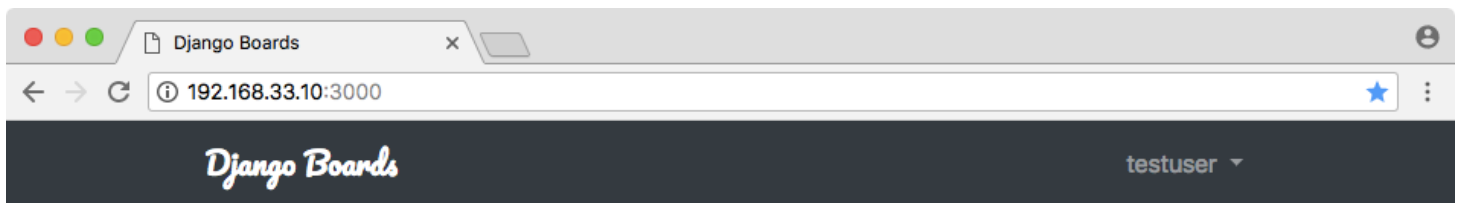
boards/views.py

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

class BoardListView(ListView):
    model = Board
    context_object_name = 'boards'
    template_name = 'home.html'
```

`model` にはモデル名、`context_object_name` にはテンプレートで参照する際の名前、`template_name` にはテンプレート名を指定する。

これで、これまでと同じ画面が表示されるようになる。



Boards			
Board	Posts	Topics	Last Post
Django Django discussion board.	4	3	By testuser at 2018年3月26日22:22
Python General discussion about Python.	2	2	By testuser at 2018年3月26日22:21
Random Here you can discuss about anything	1	1	By testuser at 2018年3月26日22:22
New test board Empty board just for testing.	0	0	No posts yet.

また、`boards/tests/test_view_home.py` テストは以下のように変更する（変更前は `self.assertEqual(view.func, home)` ）。

```
from django.test import TestCase
from django.urls import resolve
from ..views import BoardListView

class HomeTests(TestCase):
    # ...
    def test_home_url_resolves_home_view(self):
        view = resolve('/')
        self.assertEqual(view.func.view_class, BoardListView)
```

対話型シェルでページネーションの確認

リスト表示では数が多い場合にページネーションを使用するが、ListViewではページネーションを簡単に実装できる。

実装を進める前にまず是对話型シェルでページネーションの基本を確認してみる。

ページネーションの確認にはデータ数が必要なので最初に `python manage.py shell` で対話型シェルを起動してデータを追加する。

```

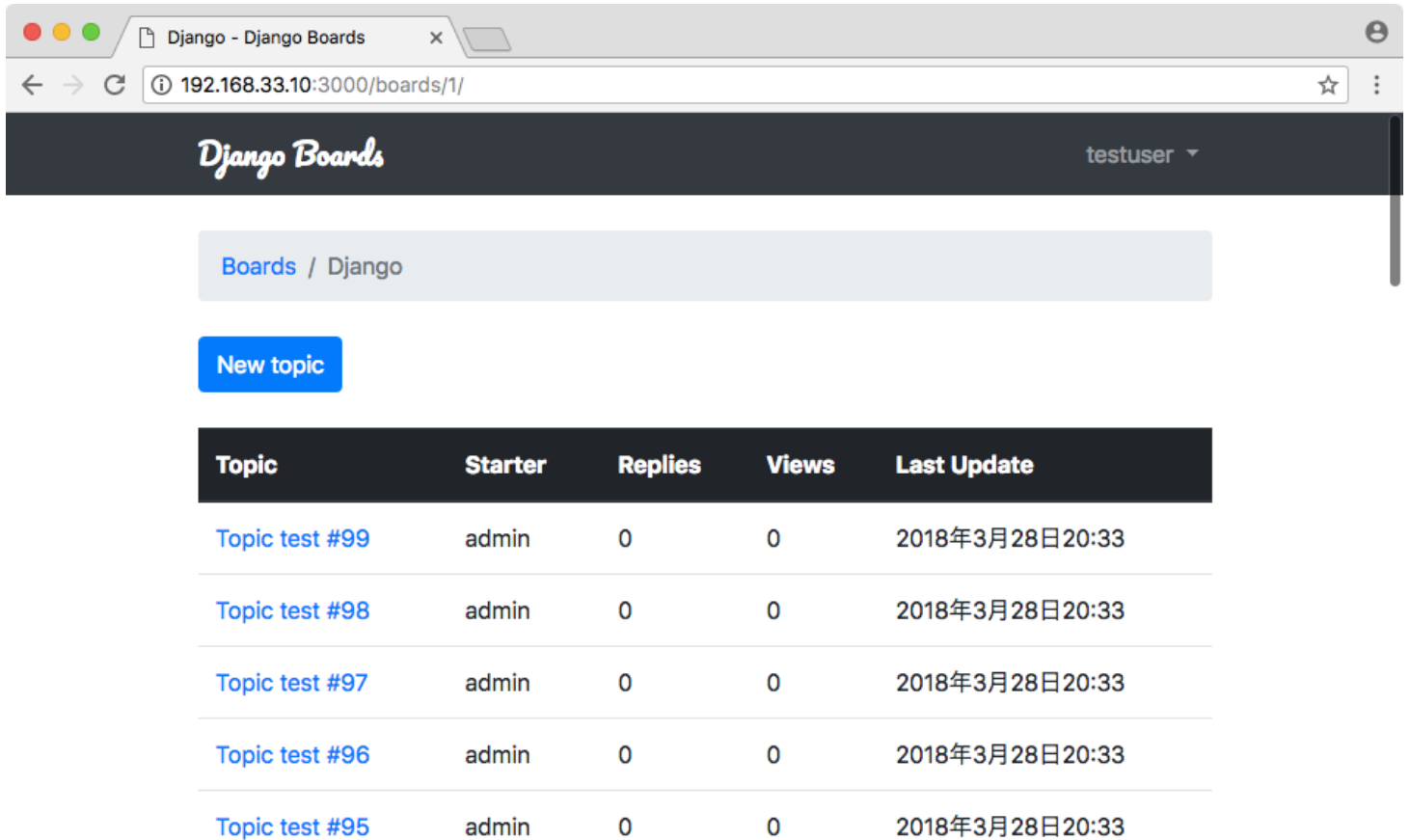
from django.contrib.auth.models import User
from boards.models import Board, Topic, Post

user = User.objects.first()
board = Board.objects.get(name='Django')

for i in range(100):
    subject = 'Topic test #{}'.format(i)
    topic = Topic.objects.create(subject=subject, board=board, starter=user)
    Post.objects.create(message='Message test...', topic=topic, created_by=user)

```

これで以下のようにデータが追加される。



The screenshot shows a web browser window with the URL `192.168.33.10:3000/boards/1/`. The page title is "Django Boards" and the user is logged in as "testuser". The breadcrumb is "Boards / Django". There is a "New topic" button. Below it is a table with the following data:

Topic	Starter	Replies	Views	Last Update
Topic test #99	admin	0	0	2018年3月28日20:33
Topic test #98	admin	0	0	2018年3月28日20:33
Topic test #97	admin	0	0	2018年3月28日20:33
Topic test #96	admin	0	0	2018年3月28日20:33
Topic test #95	admin	0	0	2018年3月28日20:33

それではページネーションについて確認してみる。なお、ページネーションの詳細は[ドキュメント](#)を参照。

```

In [1]: from boards.models import Topic

# 全Topicの数。既に登録していたデータもあるので100以上ある。
In [2]: Topic.objects.count()
Out[2]: 106

# 'Django' Boardに属するTopicの数。
In [3]: Topic.objects.filter(board__name='Django').count()
Out[3]: 103

# QuerySetを取得する。
In [4]: queryset = Topic.objects.filter(board__name='Django').order_by('-last_updated')

```

```
In [5]: from django.core.paginator import Paginator

# Paginatorクラス。QuerySetと1ページに表示するアイテム数を渡す。
In [6]: paginator = Paginator(queryset, 20)

# QuerySetのcount()と同じ
In [7]: paginator.count
Out[7]: 103

# ページ数。103÷20で6ページ。最後のページのアイテム数は3つ。
In [8]: paginator.num_pages
Out[8]: 6

# ページのレンジ。
In [9]: paginator.page_range
Out[9]: range(1, 7)

# ページのインスタンスを返す。
In [10]: paginator.page(2)
Out[10]: <Page 2 of 6>

In [11]: page = paginator.page(2)

In [12]: type(page)
Out[12]: django.core.paginator.Page

In [13]: type(paginator)
Out[13]: django.core.paginator.Paginator

# 存在しないページを指定すると例外が発生。
In [14]: paginator.page(7)
EmptyPage: That page contains no results

# 数値以外も例外が発生。
In [15]: paginator.page('abc')
PageNotAnInteger: That page number is not an integer

In [16]: page = paginator.page(1)

# 次のページがあるか。
In [17]: page.has_next()
Out[17]: True

# 前のページがあるか。
In [18]: page.has_previous()
Out[18]: False

# 次のページまたは前のページがあるか。
In [19]: page.has_other_pages()
Out[19]: True

# 次のページの番号
In [20]: page.next_page_number()
Out[20]: 2

# 前のページの番号。存在しない場合は例外が発生。
In [21]: page.previous_page_number()
EmptyPage: That page number is less than 1
```

関数ベースビューでのページネーション

ページネーションの使い方はなんとなくわかったので関数ベースビューでTopic一覧ページを実装してみる。

`boards/views.py` を以下のように編集。

```
from django.db.models import Count
from django.core.paginator import Paginator, EmptyPage, PageNotAnInteger
from django.shortcuts import get_object_or_404, render
from django.views.generic import ListView
from .models import Board

def board_topics(request, pk):
    board = get_object_or_404(Board, pk=pk)
    queryset = board.topics.order_by('-last_updated').annotate(replies=Count('posts') - 1)
    page = request.GET.get('page', 1)

    paginator = Paginator(queryset, 20)

    try:
        topics = paginator.page(page)
    except PageNotAnInteger:
        # fallback to the first page
        topics = paginator.page(1)
    except EmptyPage:
        # probably the user tried to add a page number
        # in the url, so we fallback to the last page
        topics = paginator.page(paginator.num_pages)

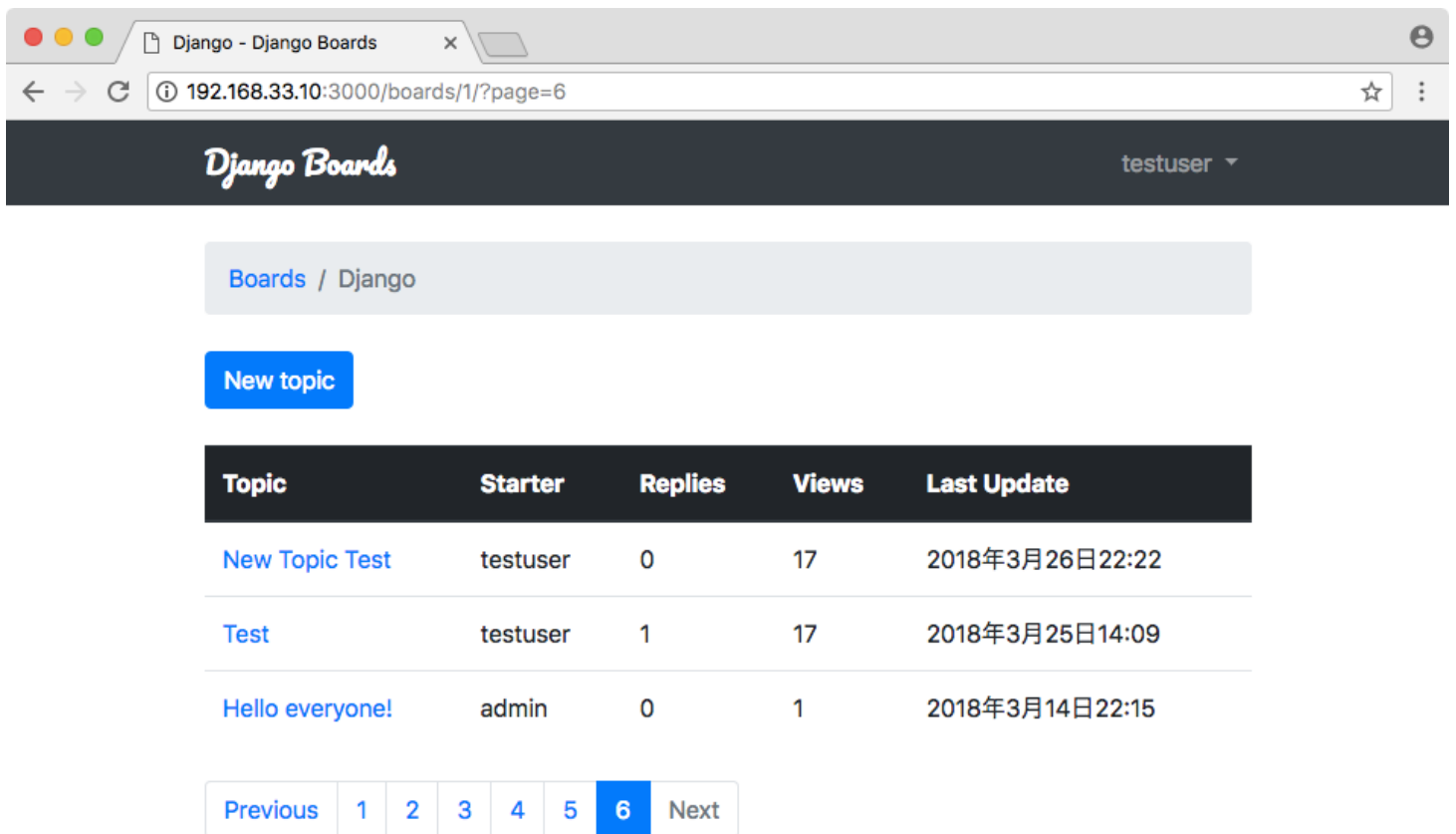
    return render(request, 'topics.html', {'board': board, 'topics': topics})
```

ここで `topics` はQuerySetではなく、ページのインスタンスになっていることに注意。

テンプレートではBootstrap 4の[Paginationコンポーネント](#)を用いて以下のようにページネーションを実現する。

▶ `templates/topics.html` (クリックで展開)

問題がなければ以下のページネーションが表示される。



クラスベース汎用ビューのListViewでのページネーション

今回はListViewを使ってTopic一覧ページのページネーションを実装してみる。

`boards/views.py` を以下のように書き換える。

```
class TopicListView(ListView):
    model = Topic
    context_object_name = 'topics'
    template_name = 'topics.html'
    paginate_by = 20

    def get_context_data(self, **kwargs):
        kwargs['board'] = self.board
        return super().get_context_data(**kwargs)

    def get_queryset(self):
        self.board = get_object_or_404(Board, pk=self.kwargs.get('pk'))
        queryset = self.board.topics.order_by('-last_updated').annotate(replies=Count('posts') - 1)
        return queryset
```

ListViewを使うとテンプレート側では `paginator` , `page_obj` , `is_paginated` , `object_list` という変数が見える。 `object_list` については `context_object_name` で別名を付けることもできる。

`paginate_by = 20` で1ページに表示するアイテム数を指定し、`get_context_data()` ではコンテキストデータに `board` を追加するためにオーバーライドしている。同様に `get_queryset()` もオーバーライドしている（オーバーライドしないと `all()` になる）。

次に `myproject/urls.py` をクラスベースビューの書き方に変更。

```
from django.conf.urls import url
from boards import views

urlpatterns = [
    # ...
    # path('boards/<int:pk>/', views.board_topics, name='board_topics'),
    path('boards/<int:pk>/', views.TopicListView.as_view(), name='board_topics'),
]
```

テンプレートは以下の通り。

▶ `templates/topics.html` （クリックで展開）

`boards/tests/test_view_board_topics.py` テストも書き換えておく。

```
from django.test import TestCase
from django.urls import resolve
from ..views import TopicListView

class BoardTopicsTests(TestCase):
    # ...
    def test_board_topics_url_resolves_board_topics_view(self):
        view = resolve('/boards/1/')
        self.assertEqual(view.func.view_class, TopicListView)
```

上記のように変更するとListViewによりページネーションが実現できる（結果は関数ベースビューの場合と同じ）。

ページネーションのテンプレートを再利用

ページネーションを毎回記述するのは面倒なので再利用できるようにしておく。

Post一覧ページをページネーションできるように `boards/views.py` を編集。ページネーションが確認しやすいように `paginate_by = 2` とする。

```
class PostListView(ListView):
    model = Post
    context_object_name = 'posts'
    template_name = 'topic_posts.html'
    paginate_by = 2
```

```

def get_context_data(self, **kwargs):
    self.topic.views += 1
    self.topic.save()
    kwargs['topic'] = self.topic
    return super().get_context_data(**kwargs)

def get_queryset(self):
    self.topic = get_object_or_404(Topic, board__pk=self.kwargs.get('pk'), pk=self.kwargs.get('t

    queryset = self.topic.posts.order_by('created_at')
    return queryset

```

`myproject/urls.py` も編集。

```

from django.conf.urls import url
from boards import views

urlpatterns = [
    # ...
    #path('boards/<int:pk>/topics/<int:topic_pk>/', views.topic_posts, name='topic_posts'),
    path('boards/<int:pk>/topics/<int:topic_pk>/', views.PostListView.as_view(), name='topic_posts')
]

```

再利用可能なページネーションのテンプレートを `templates/includes` ディレクトリに `pagination.html` として作成する。

```

└─ templates/
   └─ includes/
      └─ form.html
      └─ pagination.html

```

`templates/includes/pagination.html`

```

{% if is_paginated %}
<nav aria-label="Topics pagination" class="mb-4">
  <ul class="pagination">
    {% if page_obj.has_previous %}
      <li class="page-item">
        <a class="page-link" href="?page={{ page_obj.previous_page_number }}">Previous</a>
      </li>
    {% else %}
      <li class="page-item disabled">
        <span class="page-link">Previous</span>
      </li>
    {% endif %}

    {% for page_num in paginator.page_range %}
      {% if page_obj.number == page_num %}
        <li class="page-item active">
          <span class="page-link">
            {{ page_num }}
            <span class="sr-only">(current)</span>
          </span>

```



```

        </li>
    {% else %}
        <li class="page-item">
            <a class="page-link" href="?page={{ page_num }}">{{ page_num }}</a>
        </li>
    {% endif %}
{% endfor %}

{% if page_obj.has_next %}
    <li class="page-item">
        <a class="page-link" href="?page={{ page_obj.next_page_number }}">Next</a>
    </li>
{% else %}
    <li class="page-item disabled">
        <span class="page-link">Next</span>
    </li>
{% endif %}
</ul>
</nav>
{% endif %}

```

topic_posts.html ではページネーションのテンプレートを {% include 'includes/pagination.html' %} でインクルード。

▶ templates/topic_posts.html (クリックで展開)

topics.html も同様に書き換える。

▶ templates/topics.html (クリックで展開)

boards/tests/test_view_topic_posts.py テストの書き換えも忘れないこと。

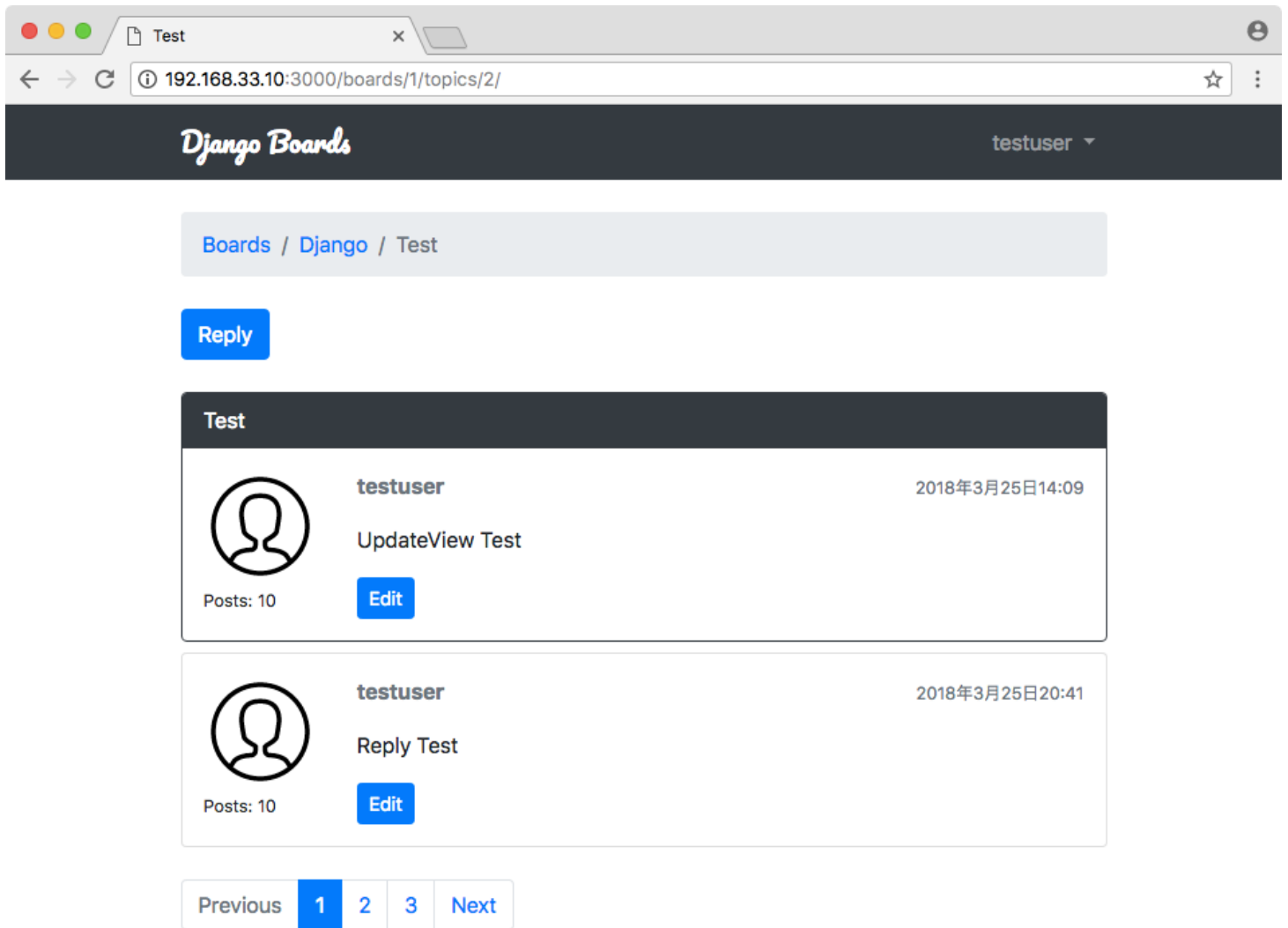
```

from django.test import TestCase
from django.urls import resolve
from ..views import PostListView

class TopicPostsTests(TestCase):
    # ...
    def test_view_function(self):
        view = resolve('/boards/1/topics/1/')
        self.assertEqual(view.func.view_class, PostListView)

```

これでPost一覧にもページネーションが追加される。



まとめ

- クラスベース汎用ビューのListViewで一覧ページが作成可能
- `Paginator` クラスでページネーション
- 独自の処理を追加したい場合は `get_context_data` , `get_queryset` をオーバーライド
- ページネーションのテンプレートは再利用できるようにしておく と 便利