3-6. ユーザー情報変更画面を作成する

2019年4月29日　コメントをする

今回のテーマは「ユーザー情報変更画面を作成する」です。ここまでログイン、ログアウト、パスワード変更機能を見てきました。このままパスワードリセット機能を実装したいところですが、ユーザーにEmailを登録する必要がありますので、一度標準のビューから離れてユーザー情報変更画面を作成しましょう。

※本ページは「3-5. PasswordChangeViewを使用してパスワード変更画面を作成する」まで読まれた方を対象としています。そのためサンプルソースコードが省略されている場合があります

ユーザー情報変更画面のフォーム作成

まずはフォームを作成しましょう。accounts/forms.pyを作成します。

accounts/forms.py

```
from django.forms import ModelForm
from django.contrib.auth.models import User

class UserChangeForm(ModelForm):
    class Meta:
        model = User
        fields = [
            'email',
            'last_name',
            'first_name',
        ]

    def __init__(self, email=None, first_name=None, last_name=None, *args, **kwargs):
        kwargs.setdefault('label_suffix', '')
        super().__init__(*args, **kwargs)
        # ユーザーの更新前情報をフォームに挿入
        if email:
            self.fields['email'].widget.attrs['value'] = email
        if first_name:
            self.fields['first_name'].widget.attrs['value'] = first_name
        if last_name:
            self.fields['last_name'].widget.attrs['value'] = last_name

    def update(self, user):
        user.email = self.cleaned_data['email']
        user.first_name = self.cleaned_data['first_name']
        user.last_name = self.cleaned_data['last_name']
        user.save()
```

UserChangeFormのインスタンス生成時にユーザー情報を渡してフォームに予め表示するよう__init__関数をオーバーライドしています。また、ユーザー情報更新用の関数としてupdate関数を定義しています。このupdate関数はModelFormの関数ではないことに注意して下さい。

ユーザー情報変更画面のビュー作成

では次にビューを作成していきましょう。

accounts/views.py(一部抜粋)

```
from django.views.generic import FormView

class UserChangeView(LoginRequiredMixin, FormView):
    template_name = 'registration/change.html'
    form_class = UserChangeForm
    success_url = reverse_lazy('accounts:profile')

    def form_valid(self, form):
        #formのupdateメソッドにログインユーザーを渡して更新
        form.update(user=self.request.user)
        return super().form_valid(form)

    def get_form_kwargs(self):
        kwargs = super().get_form_kwargs()
        # 更新前のユーザー情報をkwargsとして渡す
        kwargs.update({
            'email' : self.request.user.email,
            'first_name' : self.request.user.first_name,
            'last_name' : self.request.user.last_name,
        })
        return kwargs
```

普通のFormViewの使い方なので特段解説は不要かと思いますが、get_form_kwargs関数をオーバーライドしている部分に触れておきます。get_form_kwargs関数はdjango.views.generic.edit.FormMixinクラスの関数です。この関数が返すdict型オブジェクトがフォームクラス生成時にコンストラクタの実引数として投入されます。今回のケースですとUserChangeFormの__init__関数に渡されることになります。これにより現在のユーザー情報をフォームに表示することが可能となります。

get_form_kwargs関数をオーバーライドするとフォームにパラメータを渡すことができて便利なので覚えておくとフォームの使いみちが広がるかも知れません。

ユーザー情報変更画面のテンプレート作成

```
{% extends 'base/base.html' %}
{% block title %}ITについて切磋琢磨する掲示板 - {{ block.super }}{% endblock %}
{% block content %}
<div class="ui grid stackable">
    <div class="eleven wide column">
        <div class="ui breadcrumb">
            <a href="{% url 'base:top' %}" class="section">TOP</a>
            <i class="right angle icon divider"></i>
            <a href="{% url 'accounts:profile' %}" class="section">プロフィール</a>
            <i class="right angle icon divider"></i>
            <div class="active section">ユーザー情報の変更</div>
        </div>
        <div class="ui segment">
            <div class="content">
                <div class="header"><h3>ユーザー情報の変更</h3></div>
                <form class="ui form" action="" method="POST">
                    {% csrf_token %}
                    {{form.as_p}}
                    <button class="ui orange button" type="submit">確認</button>
                </form>
            </div>
        </div>
        <a href="{% url 'accounts:profile' %}">プロフィールに戻る</a>
    </div>
    {% include 'base/sidebar.html' %}
</div>
{% endblock %}
```

formを表示するだけなので、特段解説は不要だと思います。手を抜いてform.as_pで表示しています。

URLの設定

続いてURLの設定を行います。
accounts/urls.py(一部抜粋)

```
  urlpatterns = [
      # path('', include('django.contrib.auth.urls')),
      path('create/', views.UserCreateView.as_view(), name="create"),
      path('profile/', views.UserProfileView.as_view(), name="profile"),
+     path('change/', views.UserChangeView.as_view(), name="change"),
  ]
```

プロフィール画面のリンク修正

プロフィール画面のリンクも修正しておきましょう。

```
- <a class="ui button" href="">登録情報変更</a>
+ <a class="ui button" href="{% url 'accounts:change' %}">登録情報変更</a>
```

動作確認

では動作確認をしましょう。プロフィール画面から「登録情報変更」を押してユーザー情報変更画面に遷移して、フォームからユーザー情報を変更できればOKです。

最後に

では、次回はパスワードを忘れたユーザーのためのパスワードリセット画面について見ていきたいと思います。

3-9. LogoutViewで作成したログアウト画面をカスタマイズする

2019年4月29日　コメントをする

今回のテーマは「ログアウト画面をカスタマイズする」です。ここまではログイン画面と同様にログアウト画面も修正していきます。今回もLogoutViewを使用するケースとLogoutViewを継承するクラスを作成するケースとで分けたいと思います。

※本ページは「LoginViewで作成したログイン画面をカスタマイズする」まで読まれた方を対象としています。そのためサンプルソースコードが省略されている場合があります。

URLのカスタマイズ

もしURLが修正されていない場合はLoginViewで作成したログイン画面をカスタマイズするの「URLのカスタマイズ」を行って下さい。

ケース１：LooutViewを使用してas_view()でプロパティを渡す

このケースではビューを自作する必要はありません。as_view関数でLogoutViewのクラス変数を書き換える方法です。

accounts/urls.py(一部抜粋)

```
  urlpatterns = [
      # copy from django.contrib.auth.urls.py
      path('login/', views.CustomLoginView.as_view(), name='login'),
-     path('logout/', av.LogoutView.as_view(), name='logout'),
+     path('logout/', av.LogoutView.as_view(
+         template_name='regstration/logged_out.html',
+         next_page='/'
+     ), name='logout'),

      path('password_change/', av.PasswordChangeView.as_view(), name='password_change'),
      path('password_change/done/', av.PasswordChangeDoneView.as_view(), name='password_ch

      path('password_reset/', av.PasswordResetView.as_view(), name='password_reset'),
      path('password_reset/done/', av.PasswordResetDoneView.as_view(), name='password_rese
      path('reset///', av.PasswordResetConfirmView.as_view(), name='password_reset_confirm
      path('reset/done/', av.PasswordResetCompleteView.as_view(), name='password_reset_com

      path('create/', views.UserCreateView.as_view(), name="create"),
      path('profile/', views.UserProfileView.as_view(), name="profile"),
      path('change/', views.UserChangeView.as_view(), name="change"),
  ]
```

今回はform_classのみ書き換えましたがtemplate_name等も書き換え可能です。LoginViewの詳細は「」をご覧ください。

ケース 2 ：LogoutViewを継承するビュークラスを作る

次にLoginViewを継承して新しいビュークラスを作る場合を考えましょう。as_view関数でプロパティを書き換える方法よりも応用の効く方法です。accounts/views.pyに書き加えていきます。

accounts/views.py(一部抜粋)

```
# importはページトップ
- from django.contrib.auth.views import LoginView
+ from django.contrib.auth.views import LoginView, LogoutView

+ class CustomLogoutView(LogoutView):
+     template_name = 'registration/logged_out.html'
+     next_page = '/'
```

accounts/urlsも書き換えます。
accounts/urls.py(一部抜粋)

```
  urlpatterns = [
      # copy from django.contrib.auth.urls.py
      path('login/', views.CustomLoginView.as_view(), name='login'),
-     path('logout/', av.LogoutView.as_view(
-         template_name='regstration/logged_out.html',
-         next_page='/'
-     ), name='logout'),
+     path('logout/', views.CustomLogoutView.as_view(), name='logout'),

      path('password_change/', av.PasswordChangeView.as_view(), name='password_change'),
      path('password_change/done/', av.PasswordChangeDoneView.as_view(), name='password_ch

      path('password_reset/', av.PasswordResetView.as_view(), name='password_reset'),
      path('password_reset/done/', av.PasswordResetDoneView.as_view(), name='password_rese
      path('reset///', av.PasswordResetConfirmView.as_view(), name='password_reset_confirm
      path('reset/done/', av.PasswordResetCompleteView.as_view(), name='password_reset_com

      path('create/', views.UserCreateView.as_view(), name="create"),
      path('profile/', views.UserProfileView.as_view(), name="profile"),
      path('change/', views.UserChangeView.as_view(), name="change"),
  ]
```

ログアウト後の遷移先を変更する

ログアウト後の遷移先はLogoutViewのnext_page変数を指定するかsettigs.pyのLOGOUT_REDIRECT_URLを指定することで設定出来ます。また、GETもしくはPOSTメソッドで'next'キーのパラメータを指定しても遷移します。優先順位としては

1. GETもしくはPOSTで指定したnextパラメータ
2. next_page
3. LOGOUT_REDIRECT_URL

の順番となります。

最後に

ログアウト画面に関してはフォームもないので作業量は少ないです。カスタマイズがすくないのでLogoutViewをそのまま用いても支障は少ないと思います。次回はパスワード変更画面をカスタマイズしていきます。

3-8. LoginViewで作成したログイン画面をカスタマイズする

2019年4月28日　コメントをする

今回のテーマは「LoginViewで作成したログイン画面をカスタマイズする」です。ここまではDjangoに組み込まれたビューを使用して認証機能を実装してきましたが、ここからはカスタマイズを加えていきます。LoginViewをベースとしたカスタマイズをすることでフォームに手を加えたり、テンプレート名を変更するなど自由なカスタマイズが出来るようになります。

※本ページは「PasswordResetViewを使用してパスワードリセット画面を作成する」まで読まれた方を対象としています。そのためサンプルソースコードが省略されている場合があります。

URLのカスタマイズ

ここまではdjango.contrib.auth.urlsのurlpatternsをincludeしただけでした。これではカスタマイズは難しいのでaccounts/urls.pyにURLを設定し直しましょう。

mysite/urls.py(一部抜粋)

```
  urlpatterns = [
      path('admin/', admin.site.urls),
-     path('accounts/', include('django.contrib.auth.urls')),
      path('accounts/', include('accounts.urls')),
      path('', include('base.urls')),
      path('thread/', include('thread.urls')),
      path('api/', include('api.urls')),
      path('search/', include('search.urls')),
      path('sitemap.xml', sitemap, {'sitemaps': sitemaps}),
  ] + static(settings.MEDIA_URL, document_root=settings.MEDIA_ROOT)
```

accounts/urls.py

```
from django.urls import path, include
from django.contrib.auth import views as av
from . import views

app_name = 'accounts'

urlpatterns = [
    # copy from django.contrib.auth.urls.py
    path('login/', av.LoginView.as_view(), name='login'),
    path('logout/', av.LogoutView.as_view(), name='logout'),

    path('password_change/', av.PasswordChangeView.as_view(), name='password_change'),
    path('password_change/done/', av.PasswordChangeDoneView.as_view(), name='password_chan

    path('password_reset/', av.PasswordResetView.as_view(), name='password_reset'),
    path('password_reset/done/', av.PasswordResetDoneView.as_view(), name='password_reset_
    path('reset///', av.PasswordResetConfirmView.as_view(), name='password_reset_confirm')
    path('reset/done/', av.PasswordResetCompleteView.as_view(), name='password_reset_compl

    path('create/', views.UserCreateView.as_view(), name="create"),
    path('profile/', views.UserProfileView.as_view(), name="profile"),
    path('change/', views.UserChangeView.as_view(), name="change"),
]
```

avはauth_viewsの頭文字をとった省略形です。この時点では前回まで実装した内容とほとんど変わりありません。これでログイン、ログアウト、パスワード変更画面等、別々に設定可能な準備が整いました。

URLショートコードの修正

以降、URLのショートコードには'accounts'が付くことに注意して下さい。例えばログイン画面へのリダイレクトはredirect(reverse_lazy('accounts:login'))となります。テンプレートのurlの表記も全て変わりますので注意してください。

templates/base/base.html(一部抜粋)

```
  <div class="right menu">
      {% if user.is_authenticated %}
      <a class="item" href="{% url 'accounts:profile' %}">ユーザー情報</a>
-     <a class="item" href="{% url 'logout' %}">ログアウト</a>
+     <a class="item" href="{% url 'accounts:logout' %}">ログアウト</a>
      {% else %}
-     <a class="item" href="{% url 'login' %}">ログイン</a>
+     <a class="item" href="{% url 'accounts:login' %}">ログイン</a>
      <a class="item" href="{% url 'accounts:create' %}">ユーザー登録</a>
      {% endif %}
  </div>
```

templates/registration/login.html(一部抜粋)

```
  <a class="ui item" href="{% url 'accounts:create' %}">ユーザー登録</a>／
- <a class="ui item" href="{% url 'password_reset' %}">パスワードを忘れた場合</a>
+ <a class="ui item" href="{% url 'accounts:password_reset' %}">パスワードを忘れた場合</a>
```

templates/registration/profile.html

```
  <a class="ui button" href="{% url 'accounts:change' %}">登録情報変更</a>
- <a class="ui button" href="{% url 'password_change' %}">パスワード変更</a>
+ <a class="ui button" href="{% url 'accounts:password_change' %}">パスワード変更</a>
```

カスタムフォームの作成

LoginViewはデフォルトでAuthenticationFormを使っていますが、これでは少々融通が利かないので自分でカスタマイズ可能なフォームクラスを作成しておきましょう。

accounts/forms.py(一部抜粋)

```
+ from django.contrib.auth.forms import AuthenticationForm
+
+ class CustomAuthenticationForm(AuthenticationForm):
+     def __init__(self, *args, **kwargs):
+         kwargs.setdefault('label_suffix', '')
+         super().__init__(*args, **kwargs)
```

もしフォームにclassを付与したいなどの場合はここで追加します。

ケース 1 ：LoginViewを使用してas_view()でプロパティを渡す

このケースではビューを自作する必要はありません。as_view関数でLoginViewのクラス変数を書き換える方法です。

accounts/urls.py

```
+ from .forms import CustomAuthenticationForm

- path('login/', av.LoginView.as_view(), name='login')
+ path('login/', av.LoginView.as_view(form_class=CustomAuthenticationForm
                                      ), name='login'),
  path('logout/', av.LogoutView.as_view(), name='logout'),
```

今回はform_classのみ書き換えましたがtemplate_name等も書き換え可能です。LoginViewの詳細は「」をご覧ください。

ケース 2 ：LoginViewを継承するビュークラスを作る

次にLoginViewを継承して新しいビュークラスを作る場合を考えましょう。as_view関数でプロパティを書き換える方法よりも応用の効く方法です。accounts/views.pyに書き加えていきます。

accounts/views.py(一部抜粋)

```
#importは行頭に追加
+ from .forms import UserChangeForm, CustomAuthenticationForm

+ class CustomLoginView(LoginView):
+     form_class = CustomAuthenticationForm
```

accounts/urlsも書き換えます。
accounts/urls.py(一部抜粋)

```
- path('login/', av.LoginView.as_view(form_class=CustomAuthenticationForm
-                                      ), name='login'),
+ path('login/', views.CustomLoginView.as_view(), name='login'),
```

ログイン後の遷移先を変更する

ログイン後の遷移については以前触れましたが、改めて説明しておきます。ログイン後の遷移先URLは'/accounts/profile/'が指定されています。これを変更するにはmysite/settings.pyでLOGIN_REDIRECT_URLを指定することで設定出来ます。例えば、ログイン後にトップページに遷移したい場合は

mysite/settings.py(一部抜粋)

```
+ LOGIN_REDIRECT_URL = '/'
```

のように追加すればOKです。

最後に

カスタマイズと言ってもさほど大変なことはありません。Djangoの予めある機能を活かしてカスタマイズをしていきましょう。次回はログアウト画面をカスタマイズしていきます。

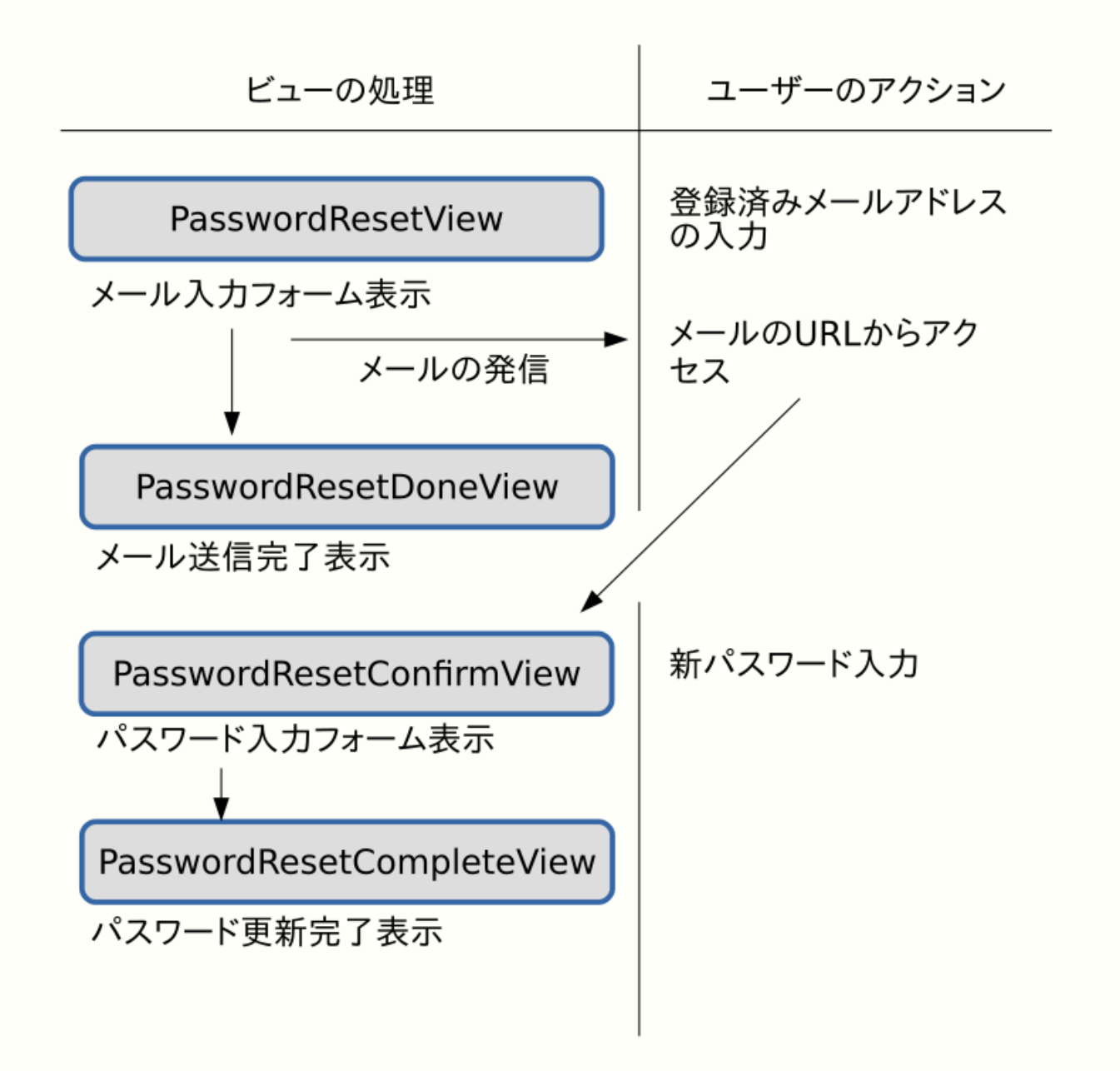
3-7. PasswordResetViewを使用してパスワードリセット画面を作成する

2019年4月25日　コメントをする

今回のテーマは「PasswordResetViewを使用してパスワードリセット画面を作成する」です。パスワードリセットとはパスワードを忘れた場合に再発行する手続きする画面のことを言います。Djangoにはこの機能用の標準ビューがあります。今回はカスタマイズはせず、標準ビューをそのまま使っていきます。

※本ページは「PasswordChangeViewを使用してパスワード変更画面を作成する」まで読まれた方を対象としています。そのためサンプルソースコードが省略されている場合があります。

パスワードリセットの仕組み
パスワードリセットに関しては４つのビューが絡み合うので少々ややこしいです。まずは概要を抑えましょう。パスワードリセットの概念図がコチラです。



この概念を理解した上で実装すると戸惑わないと思います。それでは標準ビューに４つについてそれぞれ説明していきます。

PasswordResetViewについて

パスワードリセットを開始するため、メールを発信するビューです。
django.contrib.auth.views.PasswordResetViewには以下のクラス変数があり、この値を継承クラスでオーバライドしたり、as_viewメソッドで設定することで制御することが出来ます。

- email_template_name: Emailテンプレート。デフォルトは'registration/password_reset_email.html'
- extra_email_context: Eメールテンプレートに渡す追加コンテキスト。デフォルトはNone
- form_class: デフォルトはPasswordResetForm
- from_email: Emeilのfromアドレス。デフォルトはNone
- html_email_template_name = HTML用のEmailテンプレート。デフォルトはNone
- subject_template_name: メールタイトル用テンプレート。デフォルトは'registration/password_reset_subject.txt'

- success_url: デフォルトはreverse_lazy(‘password_reset_done’)
- template_name: デフォルトは’registration/password_reset_form.html’
- title = デフォルトは’Password reset’の翻訳
- token_generator = デフォルトはdefault_token_generator

今回はすべてデフォルトの状態で使ってみます。認証機能のカスタマイズは本章の後半で扱うので、それまでお待ち下さい。

PasswordResetDoneViewについて

メール送信後はPasswordResetDoneViewが表示されます。
django.contrib.auth.views.PasswordResetDoneViewには以下のクラス変数があり、この値を継承クラスでオーバライドしたり、as_viewメソッドで設定することで制御することが出来ます。

- template_name: デフォルトは’registration/password_reset_done.html’
- title: デフォルトは’Password reset sent’の翻訳（言語がjaの場合は日本語訳)

このビューはメール送信したことをユーザーに伝えるだけの画面であり、ほぼ機能はTemplateViewと同様と考えればいいと思います。

PasswordResetConfirmViewについて

送信されたメールに記載されているURLからアクセスした場合にPasswordResetConfirmViewが表示されます。django.contrib.auth.views.PasswordResetConfirmViewには以下のクラス変数があり、この値を継承クラスでオーバライドしたり、as_viewメソッドで設定することで制御することが出来ます。

- form_class: デフォルトはSetPasswordForm
- post_reset_login: デフォルトはFalse
- post_reset_login_backend: デフォルトはNone
- success_url: デフォルトはreverse_lazy(‘password_reset_complete’)
- template_name: デフォルトは’registration/password_reset_confirm.html’
- title: デフォルトは’Enter new password’の翻訳
- token_generator: デフォルトはdefault_token_generator

このビューに関しても今回はカスタマイズせずそのまま使います。post_reset_loginについて簡単に解説しておきます。この値は新パスワードを登録した時にログインをするかどうかです。この値はデフォルトではFalseですが、Trueにしておくと新パスワードでログインしリダイレクトします。

PasswordResetCompleteViewについて

パスワードのリセット処理が無事に終了した場合にPasswordResetCompleteViewが表示されます。
django.contrib.auth.views.PasswordResetCompleteViewには以下のクラス変数があり、この値を継承クラスでオーバライドしたり、as_viewメソッドで設定することで制御することが出来ます。

- template_name: デフォルトは’registration/password_reset_complete.html’
- title: デフォルトは’Password reset complete’の翻訳

このビューはあくまでリセットが完了したことを伝える画面ですので、PasswordResetConfirmViewでsuccess_urlを別画面に指定した場合はなくても問題ありません。

パスワードリセット画面用のテンプレート作成

ではパスワードリセット画面用のテンプレートを作成していきます。
django.contrib.auth.views.PasswordChangeViewのデフォルトテンプレート
は'registration/password_change_form.html'ですので、その名前で作成していきます。

templates/registration/password_reset_form.html

```
{% extends 'base/base.html' %}
{% block title %}ITについて切磋琢磨する掲示板 - {{ block.super }}{% endblock %}
{% block content %}
<div class="ui grid stackable">
    <div class="eleven wide column">
        <div class="ui breadcrumb">
            <a class="section" href="{% url 'base:top' %}">TOP</a>
            <i class="right angle icon divider"></i>
            <div class="active section">パスワードリセット</div>
        </div>
        <div class="ui segment">
            <div class="content">
                <div class="header"><h3>パスワードリセット</h3></div>
                <form class="ui form" method="POST" action="">
                    {% csrf_token %}
                    {{form.as_p}}
                    <button type="submit" class="ui orange button">メールを送信する</buttor
                </form>
            </div>
        </div>
    </div>
    {% include 'base/sidebar.html' %}
</div>
{% endblock %}
```

次にPasswordResetDoneViewのテンプレートを準備していきます。
templates/registration/password_reset_done.html

```
{% extends 'base/base.html' %}
{% block title %}ITについて切磋琢磨する掲示板 - {{ block.super }}{% endblock %}
{% block content %}
<div class="ui grid stackable">
    <div class="eleven wide column">
        <div class="ui breadcrumb">
            <a class="section" href="{% url 'base:top' %}">TOP</a>
            <i class="right angle icon divider"></i>
            <div class="active section">パスワード変更</div>
        </div>
        <div class="ui segment">
            <div class="content">
                <div class="content">
                    <div class="header"><h3>パスワードリセット</h3></div>
                    <p>パスワードリセット手続きのためメールを送信しました。メールを確認の上
                </div>
            </div>
        </div>
    </div>
    {% include 'base/sidebar.html' %}
</div>
{% endblock %}
```

次にPasswordResetConfirmView用のテンプレートを作成していきます。
templates/registration/password_reset_confirm.html

```
{% extends 'base/base.html' %}
{% block title %}ITについて切磋琢磨する掲示板 - {{ block.super }}{% endblock %}
{% block content %}
<div class="ui grid stackable">
    <div class="eleven wide column">
        <div class="ui breadcrumb">
            <a class="section" href="{% url 'base:top' %}">TOP</a>
            <i class="right angle icon divider"></i>
            <div class="active section">パスワードリセット</div>
        </div>
        <div class="ui segment">
            <div class="content">
                <div class="header"><h3>パスワードリセット</h3></div>
                <form class="ui form" method="POST" action="">
                    {% csrf_token %}
                    {{form.as_p}}
                    <button type="submit" class="ui orange button">変更する</button>
                </form>
            </div>
        </div>
    </div>
    {% include 'base/sidebar.html' %}
</div>
{% endblock %}
```

次にPasswordResetConfirmView用のテンプレートを作成していきます。
templates/registration/password_reset_complete.html

```
{% extends 'base/base.html' %}
{% block title %}ITについて切磋琢磨する掲示板 - {{ block.super }}{% endblock %}
{% block content %}
<div class="ui grid stackable">
    <div class="eleven wide column">
        <div class="ui breadcrumb">
            <a class="section" href="{% url 'base:top' %}">TOP</a>
            <i class="right angle icon divider"></i>
            <div class="active section">パスワードリセット完了</div>
        </div>
        <div class="ui segment">
            <div class="content">
                <div class="content">
                    <div class="header"><h3>パスワードリセットが完了しました</h3></div>
                    <a href="{% url 'base:top' %}">TOPに戻る</a>
                </div>
            </div>
        </div>
    </div>
    {% include 'base/sidebar.html' %}
</div>
{% endblock %}
```

URLリンクの修正

ログイン画面のリンクも修正しておきましょう。
templates/registration/login.html(一部抜粋)

```
  <a class="ui item" href="{% url 'accounts:create' %}">ユーザー登録</a>／
- <a class="ui item" href="">パスワードを忘れた場合</a>
+ <a class="ui item" href="{% url 'password_reset' %}">パスワードを忘れた場合</a>
```

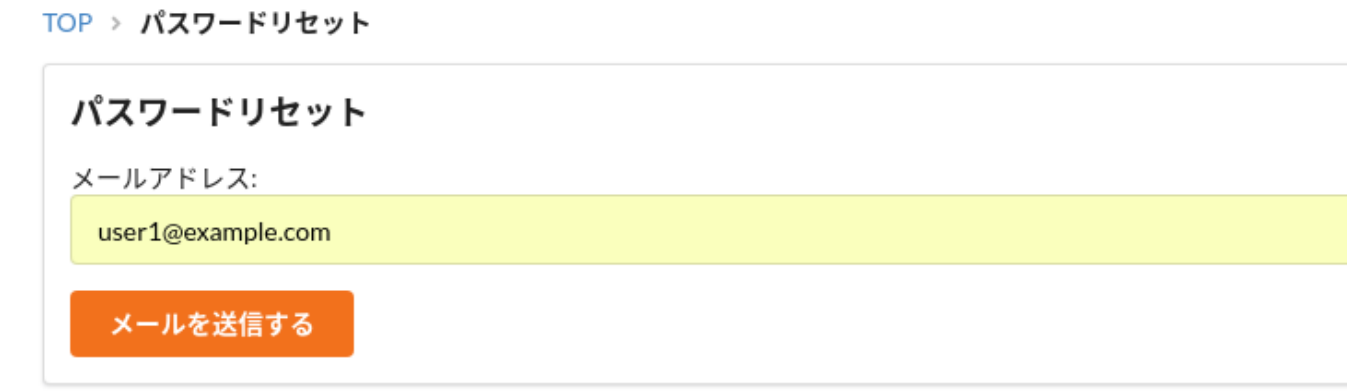
urlのショートカットが'password_reset'であることに注意して下さい。

動作確認

では動作を確認してみましょう。プロフィール画面から「パスワード変更」を押してパスワード変更画面を表示パスワードを変更してみます。パスワード変更成功の画面が出ればOKです。念の為パスワードが変更されているか再度ログインして確かめてみましょう。

※動作確認前に必ずLoginViewを使用してログイン画面を作成するで紹介した要領でmysite/urls.pyの変更をしておいて下さい。

パスワードリセット開始



メール送信済みの画面表示

送信されたメールを確認

このメールは mysite で、あなたのアカウントのパスワードリセットが要求されたため、送信されました。

次のページで新しいパスワードを選んでください:

http://mydomain.com/accounts/reset/NA/55b-e6a3b38d787906755e98/

あなたのユーザー名 (念のため): user1

ご利用ありがとうございました！

mysite チーム

送信されたメールからパスワードリセット画面にアクセスしパスワード変更

パスワードリセット完了画面

メールテンプレートの作成

さて、送信されたメールの文言どうでしょうか？ちょっとこのまま使うのは難しい気がしますよね。ではメールテンプレートを変更していきましょう。
templates/registration/password_reset_complete.html

```
{{ user.username }} 様

下記URLよりサイトにアクセスの上、パスワードの再設定を行ってください。

再設定用URL
{{ protocol }}://{{ domain }}{% url 'password_reset_confirm' uidb64=uid token=token %}

本メールは{{protocol}}://{{domain}}より自動送信されています。
心当たりのない場合は破棄をお願いします。
```

あと、メールタイトルも変更しておきましょう。これも組込みテンプレートと同名のファイルを作成することで解決します。拡張子がtxtなことに注意してくださいね。
templates/registration/password_reset_subject.txt

```
IT学習ちゃんねるのパスワードリセットのお知らせ
```

これでパスワードリセット処理時に送信されるメールは以下のようになります。

最後に

既存のビューを使用しているのでコーディング作業のほとんどがテンプレートを作るだけの簡単な作業ですね。では次回から組込みの認証用ビューを利用したカスタマイズに入っていきます。

3-5. PasswordChangeViewを使用してパスワード変更画面を作成する

2019年4月24日　コメントをする

今回のテーマは「パスワード変更画面を作成する」です。Djangoにはパスワード変更専用のPasswordChangeViewというビューが予め用意されています。今回はPasswordChangeViewを中心に見ていきます。

PasswordChangeViewについて

django.contrib.auth.views.PasswordChangeViewには以下のクラス変数があり、この値を継承クラスでオーバライドしたり、as_viewメソッドで設定することで制御することが出来ます。

- form_class: デフォルトはPasswordChangeForm
- success_url: デフォルトはreverse_lazy('password_change_done')
- template_name: デフォルトは'registration/password_change_form.html'
- title: デフォルトは'Password change'の翻訳（言語がjaの場合は日本語訳)

今回はすべてデフォルトの状態で使ってみます。認証機能のカスタマイズは本章の後半で扱うので、それまでお待ち下さい。

PasswordChangeDoneViewについて

パスワード変更後はPasswordChangeDoneViewが表示されます。
django.contrib.auth.views.PasswordChangeDoneViewには以下のクラス変数があり、この値を継承クラスでオーバライドしたり、as_viewメソッドで設定することで制御することが出来ます。

- template_name: デフォルトは'registration/password_change_done.html'
- title: デフォルトは'Password change successful'の翻訳（言語がjaの場合は日本語訳)

PasswordChangeDoneViewはログイン時しかアクセスできない等の制限はありますが、ほぼTemplateViewと同等のものだと考えて問題ないと思います。

パスワード変更画面用のテンプレート作成

ではパスワード変更画面用のテンプレートを作成していきます。
django.contrib.auth.views.PasswordChangeViewのデフォルトテンプレートは'registration/password_change_form.html'ですので、その名前で作成していきます。

templates/registration/password_change_form.html

```
{% extends 'base/base.html' %}
{% block title %}ITについて切磋琢磨する掲示板 - {{ block.super }}{% endblock %}
{% block content %}
<div class="ui grid stackable">
    <div class="eleven wide column">
        <div class="ui breadcrumb">
            <a class="section" href="{% url 'base:top' %}">TOP</a>
            <i class="right angle icon divider"></i>
            <div class="active section">パスワード変更</div>
        </div>
        <div class="ui segment">
            <div class="content">
                <div class="header"><h3>パスワード変更</h3></div>
                <form class="ui form" method="POST" action="">
                    {% csrf_token %}
                    {{form.as_p}}
                    <button type="submit" class="ui orange button">変更する</button>
                </form>
            </div>
        </div>
        <a href="{% url 'accounts:profile' %}">プロフィールに戻る</a>
    </div>
    {% include 'base/sidebar.html' %}
</div>
{% endblock %}
```

次にPasswordChangeDoneView用のテンプレートを準備していきます。
templates/registration/password_change_done.html

```
{% extends 'base/base.html' %}
{% block title %}ITについて切磋琢磨する掲示板 - {{ block.super }}{% endblock %}
{% block content %}
<div class="ui grid stackable">
    <div class="eleven wide column">
        <div class="ui breadcrumb">
            <a class="section" href="{% url 'base:top' %}">TOP</a>
            <i class="right angle icon divider"></i>
            <a class="section" href="{% url 'accounts:profile' %}">プロフィール</a>
            <i class="right angle icon divider"></i>
            <div class="active section">パスワード変更</div>
        </div>
        <div class="ui segment">
            <div class="content">
                <div class="content">
                    <div class="header"><h3>パスワード変更しました</h3></div>
                    <a href="{% url 'accounts:profile' %}">プロフィールに戻る</a>
                </div>
            </div>
        </div>
    </div>
    {% include 'base/sidebar.html' %}
</div>
{% endblock %}
```

ついでにプロフィール画面のリンクも修正しておきましょう。
templates/registration/profile.html(一部抜粋)

```
  <a class="ui button" href="">登録情報変更</a>
- <a class="ui button" href="">パスワード変更</a>
+ <a class="ui button" href="{% url 'password_change' %}">パスワード変更</a>
```

urlのショートカットが'password_change'であることに注意して下さい。

URLの確認

前回設定しているのでURLに関しては設定不要ですが、本ページから見た読者のためにmysite.pyを表示しておきます。

mysite/urls.py(一部抜粋)

```
urlpatterns = [
    path('admin/', admin.site.urls),
    path('accounts/', include('django.contrib.auth.urls')),
    path('accounts/', include('accounts.urls')),
    path('', include('base.urls')),
    path('thread/', include('thread.urls')),
    path('api/', include('api.urls')),
    path('search/', include('search.urls')),
    path('sitemap.xml', sitemap, {'sitemaps': sitemaps}),
] + static(settings.MEDIA_URL, document_root=settings.MEDIA_ROOT)
```

動作確認

では動作を確認してみましょう。プロフィール画面から「パスワード変更」を押してパスワード変更画面を表示パスワードを変更してみます。パスワード変更成功の画面が出ればOKです。念の為パスワードが変更されているか再度ログインして確かめてみましょう。

プロフィール画面から「パスワード変更」でパスワード変更画面にアクセス

パスワード変更画面でパスワード変更処理

パスワード変更後に成功画面に遷移する

最後に

既存のビューを使用しているので作業としてはテンプレートを作るだけの簡単な作業ですね。次回はパスワードリセット画面について見ていきます。

3-2. ユーザープロフィール画面を作成する

2019年4月24日　コメントをする

今回のテーマは「ユーザープロフィール画面を作成する」です。ユーザー情報に関する画面はログインユーザーしかアクセスできない仕様を想定しています。ここではログインユーザーのみアクセスできるビューについて解説していきます。

※本ページは「ユーザー登録画面を作成する」まで読まれた方を対象としています。そのためサンプルソースコードが省略されている場合があります。

LoginRequiredMixinとlogin_requiredデコレータ

Djangoにはログインユーザーしかアクセスできないする仕掛けとしてlogin_requiredデコレータとLoginRequiredMixinがあります。login_requiredデコレータは関数ベースビューに用います。LoginRequiredMixinはクラスベースビューで使用します。複数のクラスを継承する際にはLoginRequiredMixinは一番先に記載するように注意してください。login_requiredもしくはLoginRequiredMixinが付与されているビューにアクセスしようとするとログイン画面に飛ばされるようになっています。

- login_requiredデコレータに関する公式ドキュメント
- LoginRequiredMixinに関する公式ドキュメント

LOGIN_URLについて

LoginRequiredMixinを継承したクラス（正確にはAccessMixin)を継承したクラスはlogin_urlを設定することが出来ます。これはログイン画面のURLです。もしこれが設定されていない場合はsettings.pyのLOGIN_URLが用いられます。デフォルトでは'/accounts/login/'です。
ちなみにlogin_requiredデコレータでは引数でlogin_urlを渡します。

ユーザープロフィール画面を作成する

まずはテンプレートを新規作成します。
templates/registration/profile.html

```
{% extends 'base/base.html' %}
{% block title %}ITについて切磋琢磨する掲示板 - {{ block.super }}{% endblock %}
{% block content %}
<div class="ui grid stackable">
    <div class="eleven wide column">
        <div class="ui breadcrumb">
            <a href="{% url 'base:top' %}" class="section">TOP</a>
            <i class="right angle icon divider"></i>
            <a class="active section">ユーザー情報</a>
        </div>
        <div class="ui segment">
            <div class="content">
                <div class="header"><h3>ユーザー情報</h3></div>
                <div class="ui divided bulleted list">
                    <div class="item">ログインID：{{user.username}}</div>
                    <div class="item">E-mail：{% if user.email %}{{user.email}}{% else %}未
                    <div class="item">名字：{% if user.last_name %}{{user.last_name}}{% el
                    <div class="item">名前：{% if user.first_name %}{{user.first_name}}{%
                </div>
            </div>
        </div>
        <a class="ui button" href="">登録情報変更</a>
        <a class="ui button" href="">パスワード変更</a>
    </div>
    {% include 'base/sidebar.html' %}
</div>
{% endblock %}
```

次にビューを作成していきます。
accounts/views.py(一部抜粋)

```
from django.contrib.auth.mixins import LoginRequiredMixin

class UserProfileView(LoginRequiredMixin, TemplateView):
    template_name = 'registration/profile.html'
    def get_queryset(self):
        return User.objects.get(id=self.request.user.id)
```

先程説明したLoginRequiredMixinが使われていますね。ここではlogin_urlは設定していないので、もしログインしない状態で、このページにアクセスすると/accounts/login/にアクセスしようとしてエラーとなります。(現段階ではログイン画面作成していないため)

では作成したプロフィール画面にアクセスするためのURLを設定しましょう。
accounts/urls.py(一部抜粋)

```
  urlpatterns = [
      # path('', include('django.contrib.auth.urls')),
      path('create/', views.UserCreateView.as_view(), name="create"),
+     path('profile/', views.UserProfileView.as_view(), name="profile"),
  ]
```

ヘッダーのURL修正

画面のヘッダーの「ユーザーの情報」にURLを設定しましょう。

```
  {% load static %}
  <!DOCTYPE html>
  <head>
      <meta charset="UTF-8">
      <meta http-equiv="content-language" content="ja">
      <meta name="viewport" content="width=device-width, initial-scale=1.0, maximum-scale=
      {% block meta_tag %}{% endblock %}
      <link href="{% static 'css/semantic.css' %}" rel="stylesheet">
      {% block css %}{% endblock %}
      <title>
          {% block title %}IT学習ちゃんねる{% endblock %}
      </title>
  </head>
  <body>
      <div class="ui stackable inverted menu">
          <a href="{% url 'base:top' %}" class="header item">
              IT学習ちゃんねる
          </a>
          <a href="{% url 'base:about' %}" class="item">
              このサイトはなに？
          </a>
          <a class="item" href="{% url 'thread:create_topic' %}">
              トピック作成
          </a>
          <div class="right menu">
              {% if user.is_authenticated %}
-             <a class="item" href="">ユーザー情報</a>
+             <a class="item" href="{% url 'accounts:profile' %}">ユーザー情報</a>
              <a class="item" href="">ログアウト</a>
              {% else %}
              <a class="item" href="">ログイン</a>
              <a class="item" href="{% url 'accounts:create' %}">ユーザー登録</a>
              {% endif %}
          </div>
      </div>

      <div class="ui container" style="min-height:100vh;">
          {% block content %}
          {% endblock %}
      </div>
      <div class="ui inverted stackable footer segment">
          <div class="ui container center aligned">
              <div class="ui horizontal inverted small divided link list">
                  <a href="{% url 'base:top' %}" class="item">© 2019 IT学習ちゃんねる(仮)</
                  <a href="{% url 'base:terms' %}" class="item">利用規約</a>
                  <a href="{% url 'base:policy' %}" class="item">プライバシーポリシー</a>
              </div>
          </div>
```

```
    </div>
    <script src="https://code.jquery.com/jquery-3.1.1.js"></script>
    <script type="text/javascript" src="{% static 'js/semantic.js' %}"></script>
    {% block js %}{% endblock %}
  </body>
```

ユーザー登録後の遷移先を変更する

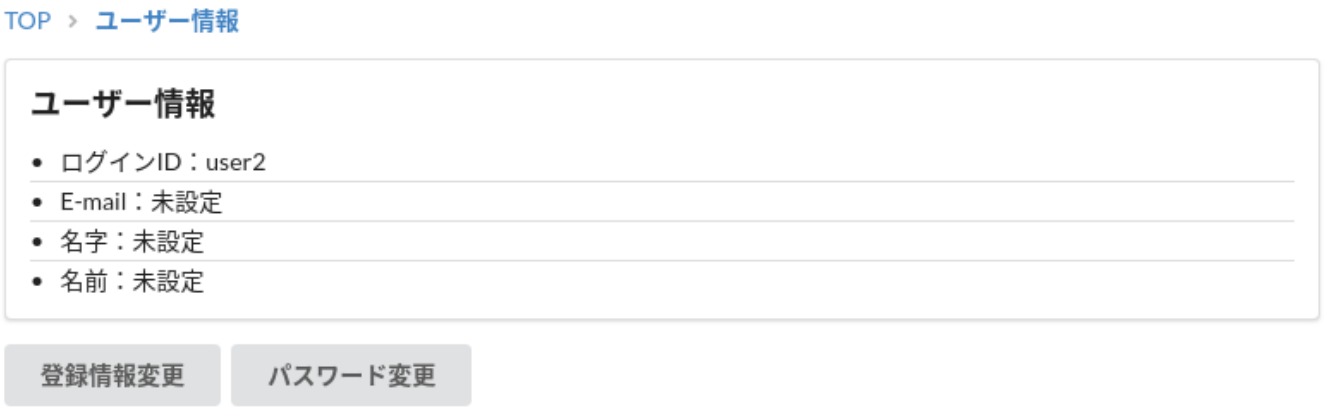
現在はユーザー登録後にトップページに遷移するようになっています。これを登録後にユーザープロフィール画面に遷移するように変更しましょう。

accounts/views.py(一部抜粋)

```
  class UserCreateView(FormView):
      form_class = UserCreationForm
      template_name = 'registration/create.html'
-     success_url = reverse_lazy('base:top')
+     success_url = reverse_lazy('accounts:profile')
      def form_valid(self, form):
          print(self.request.POST['next'])
          if self.request.POST['next'] == 'back':
              return render(self.request, 'registration/create.html', {'form': form})
          elif self.request.POST['next'] == 'confirm':
              return render(self.request, 'registration/create_confirm.html', {'form': for

          elif self.request.POST['next'] == 'regist':
              form.save()
              # 認証
              user = authenticate(
                  username=form.cleaned_data['username'],
                  password=form.cleaned_data['password1'],
              )
              # ログイン
              login(self.request, user)
              return super().form_valid(form)
          else:
              # 通常このルートは通らない
              return redirect(reverse_lazy('base:top'))
```

では確認してみましょう。ユーザーを登録すると無事にプロフィール画面に遷移すればOKです。(現時点ではログインページがないのでログインしない状態でアクセスするとエラーとなります。)



最後に

今回は認証されたユーザーのみがアクセス出来るページについて見てきました。次回はログイン処理を実装してきます。

3-4. LogoutViewを使用してログアウト機能を実装する

2019年4月23日　コメントをする

今回のテーマは「LogoutViewを使用してログアウト機能を実装する」です。前回扱ったログイン処理と対となるログアウト処理を扱います。今回もDjangoに組み込まれたLogoutViewを使用することでコーディングの手間をかけずに実装していきます。

※本ページは「LoginViewを使用してログイン画面を作成する」まで読まれた方を対象としています。そのためサンプルソースコードが省略されている場合があります

LogoutViewについて

django.contrib.auth.views.LogoutViewには以下のクラス変数があり、この値を継承クラスでオーバライドしたり、as_viewメソッドで設定することで制御することが出来ます。

- next_page: ログアウト後のリダイレクト先URL。デフォルトはNone
- redirect_field_name: POSTやGETでリダイレクト先を指定するときのキー。デフォルトはnext
- template_name: テンプレート名。デフォルトはregistration/logged_out.html
- extra_context: 追加コンテキスト。デフォルトはNone

今回はすべてデフォルトの状態で使ってみます。認証機能のカスタマイズは本章の後半で扱うので、それまでお待ち下さい。

ログアウト画面用のテンプレート作成

ではログアウト画面用のテンプレートを作成していきます。django.contrib.auth.views.LogoutViewのデフォルトテンプレートは'registration/logged_out.html'ですので、その名前で作成していきます。

templates/registration/logged_out.html

```
{% extends 'base/base.html' %}
{% block title %}ITについて切磋琢磨する掲示板 - {{ block.super }}{% endblock %}
{% block content %}
<div class="ui grid stackable">
    <div class="eleven wide column">
        <div class="ui breadcrumb">
            <a href="{% url 'base:top' %}" class="section">TOP</a>
            <i class="right angle icon divider"></i>
            <a class="active section">ログアウト完了</a>
        </div>
        <div class="ui segment">
            <div class="content">
                <div class="header"><h3>ログアウトしました。</h3></div>
                <p><a href="{% url 'base:top' %}">TOPへ戻る</a></p>
            </div>
        </div>
    </div>
    {% include 'base/sidebar.html' %}
</div>
{% endblock %}
```

URLの確認

LoginViewを使用してログイン画面を作成するの「django.contrib.auth.urlsをインクルードする」は必ず設定しておいて下さい。

前回設定しているのでURLに関しては設定不要ですが、本ページから見た読者のためにmysite.pyを表示しておきます。

mysite/urls.py(一部抜粋)

```
urlpatterns = [
    path('admin/', admin.site.urls),
    path('accounts/', include('django.contrib.auth.urls')),
    path('accounts/', include('accounts.urls')),
    path('', include('base.urls')),
    path('thread/', include('thread.urls')),
    path('api/', include('api.urls')),
    path('search/', include('search.urls')),
    path('sitemap.xml', sitemap, {'sitemaps': sitemaps}),
] + static(settings.MEDIA_URL, document_root=settings.MEDIA_ROOT)
```

動作確認

では動作を確認してみましょう。
ログインした状態でヘッダーの「ログアウト」を押してログアウト画面に遷移すればOKをです。

ヘッダーのログアウトボタン押下

ログアウト画面に遷移

最後に

さてログイン、ログアウトについては実装出来ましたね。次回はパスワード変更機能を実装していきますよ。

3-3. LoginViewを使用してログイン画面を作成する

2019年4月22日　コメントをする

今回のテーマは「LoginViewを使用してログイン画面を作成する」です。前回まではビューを自作していましたが、今回はDjangoに組み込まれたLoginViewを用いてログイン機能を構築していきます。3章の初めでも書きましたが、ログイン機能はテンプレートを作るだけでスピーディに実現できます。

※本ページは「ユーザープロフィール画面を作成する」まで読まれた方を対象としています。そのためサンプルソースコードが省略されている場合があります。

django.contrib.auth.urlsをインクルードする

Djangoの認証用のビューを使用する際のURLの設定で最も簡単なのはプロジェクト用のurls.py（今回のケースだとmysite/urls.py)にdjango.contrib.auth.urlsのurlpatternsをインクルードすることです。具体的にはmysite/urls.pyを以下のように変更します。

```
  urlpatterns = [
      path('admin/', admin.site.urls),
+     path('accounts/', include('django.contrib.auth.urls')),
      path('accounts/', include('accounts.urls')),
      path('', include('base.urls')),
      path('thread/', include('thread.urls')),
      path('api/', include('api.urls')),
      path('search/', include('search.urls')),
      path('sitemap.xml', sitemap, {'sitemaps': sitemaps}),
  ] + static(settings.MEDIA_URL, document_root=settings.MEDIA_ROOT)
```

ではここでdjango.contrib.auth.urlsのurlpatternsの中身を見てみましょう。

django/contrib/auth/urls.py(一部抜粋)

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

urlpatterns = [
    path('login/', views.LoginView.as_view(), name='login'),
    path('logout/', views.LogoutView.as_view(), name='logout'),

    path('password_change/', views.PasswordChangeView.as_view(), name='password_change'),
    path('password_change/done/', views.PasswordChangeDoneView.as_view(), name='password_c

    path('password_reset/', views.PasswordResetView.as_view(), name='password_reset'),
    path('password_reset/done/', views.PasswordResetDoneView.as_view(), name='password_res
    path('reset///', views.PasswordResetConfirmView.as_view(), name='password_reset_confir
    path('reset/done/', views.PasswordResetCompleteView.as_view(), name='password_reset_co
]
```

このurlspatternsをmysite/urls.pyにインクルードすることで以下のように各URLで各ビューにアクセスすることにあります。

/accounts/login/ -> LoginView ログイン機能
/accounts/logout/ -> LogoutView ログアウト機能
/accounts/password_change/ -> PasswordChangeView パスワード変更機能
/accounts/password_reset/ -> PasswordResetView パスワードリセット機能
※他は省略

テンプレートの作成

今回はビューに関しては組込みのLoginViewを使用するのでテンプレートを作成しましょう。registration/login.htmlを作成します。さて、ここで何故templatesディレクトリ内にaccountsディレクトリではなくregistrationディレクトリを作成したか説明します。LoginViewのクラス変数であるtemplate_nameのデフォルト値は'registration/login.html'なのです。他のLogoutViewのテンプレートのデフォルト値は'registration/logout.html'であり、他も同様です。よって統一するためにURLが/accounts/でアクセスするビューに関してはregistrationに統一しました。

templates/registration/login.html

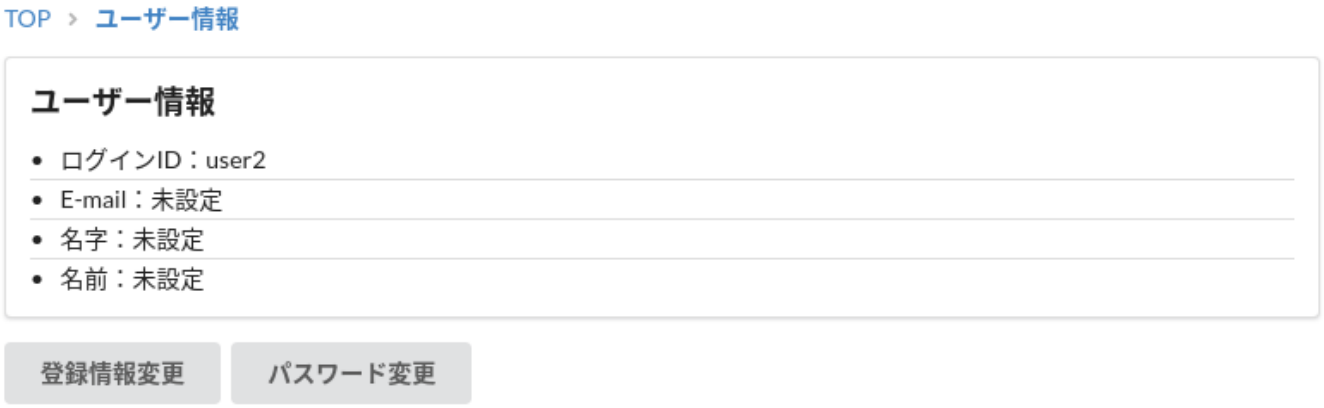
```
{% extends 'base/base.html' %}
{% block title %}ITについて切磋琢磨する掲示板 - {{ block.super }}{% endblock %}
{% block content %}
<div class="ui grid stackable">
    <div class="eleven wide column">
        <div class="ui breadcrumb">
            <a href="{% url 'base:top' %}" class="section">TOP</a>
            <i class="right angle icon divider"></i>
            <a class="active section">ログイン</a>
        </div>
        <div class="ui segment">
            <div class="content">
                <div class="header"><h3>ログイン</h3></div>
                <form class="ui form" action="" method="POST">
                    {% csrf_token %}
                    {{form.as_p}}
                    <button class="ui orange button" type="submit">ログイン</button>
                </form>
            </div>
        </div>
        <a class="ui item" href="{% url 'accounts:create' %}">ユーザー登録</a>／
        <a class="ui item" href="{% url 'password_reset' %}">パスワードを忘れた場合</a>
    </div>
    {% include 'base/sidebar.html' %}
</div>
{% endblock %}
```

それでは画面のヘッダーのURLも修正しておきましょう。
templates/registration/login.html

```
    <div class="ui stackable inverted menu">
        <a href="{% url 'base:top' %}" class="header item">
            IT学習ちゃんねる
        </a>
        <a href="{% url 'base:about' %}" class="item">
            このサイトはなに？
        </a>
        <a class="item" href="{% url 'thread:create_topic' %}">
            トピック作成
        </a>
        <div class="right menu">
            {% if user.is_authenticated %}
            <a class="item" href="{% url 'accounts:profile' %}">ユーザー情報</a>
            <a class="item" href="">ログアウト</a>
            {% else %}
-           <a class="item" href="">ログイン</a>
+           <a class="item" href="{% url 'login' %}">ログイン</a>
            <a class="item" href="{% url 'accounts:create' %}">ユーザー登録</a>
            {% endif %}
        </div>
    </div>
```

ログイン画面へのURLに関して”{% url ‘login’ %}”ですね。’accounts:login’ではないことに注意して下さい。django/contrib/auth/urls.pyではapp_nameを設定していないので、このURLのショートカットは’login’となります。

では掲示板にアクセスして確認してみましょう。localhost:8080/accounts/login/にアクセスしてログイン画面から作成したユーザーでログインできればOKです。



ログイン後の遷移先は？

さて、無事ユーザー情報ページに遷移しましたか？ところで1つの疑問が湧きます。なぜユーザー情報ページに遷移したのでしょうか？LoginViewはGETもしはPOSTで遷移先のURLが与えられた場合はそのURLに遷移し、特に指定がなければsettings.LOGIN_REDIRECT_URLに遷移します。このLOGIN_REDIRECT_URLのデフォルト値は’/accounts/profile/’なのです。よって先に作成しておいたユーザー情報ページに遷移シました。尚、GET／POSTでURLを与える時はパラメータのキーはデフォルトで”next”です。これはLoginViewのクラス変数であるredirect_field_nameで変更できます。

最後に

今回はLoginViewを中心に見てきました。次回はログアウト処理を見ていきます。

3-1. ユーザー登録画面を作成する

2019年4月19日　コメントをする

今回のテーマは「ユーザー登録画面を作成する」です。まずはユーザー登録画面を作成しましょう。これらの画面（ビュー）はDjangoでは用意されていませんので自作する必要があります。もうDjangoに慣れてきた皆さんなら簡単ですよね？

※本ページは第一章、第二章の続きとして書かれています。省略されたソースコードやプロジェクトの説明がある可能性があります。ご了承下さい。

accountsアプリケーションの作成

まずはユーザー情報を扱うaccountsアプリケーションを作成しましょう。

```
(venv)$ ./manage.py startapp accounts
```

例によってaccounts/urls.pyも作成しておきます。

accounts/urls.py

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

app_name = 'accounts'

urlpatterns = [
]
```

ひとまずurlpatternsはから配列で作成しておきます。

また、mysite/settings.pyとmysite/urls.pyにも追記をしておきます。
mysite/settings.py(一部抜粋)

```
  INSTALLED_APPS = [
      'django.contrib.admin',
      'django.contrib.auth',
      'django.contrib.contenttypes',
      'django.contrib.sessions',
      'django.contrib.messages',
      'django.contrib.staticfiles',
      'django.contrib.sites',
      'django.contrib.sitemaps',
      'debug_toolbar',
      'base',
      'thread',
      'api',
      'search',
+     'accounts',
  ]
```

mysite/settings.py(一部抜粋)

```
  urlpatterns = [
      path('admin/', admin.site.urls),
+     path('accounts/', include('accounts.urls')),
      path('', include('base.urls')),
      path('thread/', include('thread.urls')),
      path('api/', include('api.urls')),
      path('search/', include('search.urls')),
      path('sitemap.xml', sitemap, {'sitemaps': sitemaps}),
  ] + static(settings.MEDIA_URL, document_root=settings.MEDIA_ROOT)
```

UserCreationFormについて

今回はDjangoに備わっているUserCreationFormを使ってユーザー登録画面を作成していきます。このUserCreationFormはDjango標準の認証機能として搭載されているUserモデルの作成フォームです。Userをカスタマイズする場合はUserCreationFormもカスタマイズする必要が出てきますのでご注意下さい。

UserCreationFormに関しては公式ドキュメントのDjangoの認証システムを使用するをご覧ください。

ユーザー登録画面の作成

まずはユーザー登録画面を作成しましょう。特に難しいことはありません。FormViewを継承したクラスを作り、トピック登録画面と同様に確認画面付きの画面を作成していきます。ただし、今回は単にユーザーを登録するだけではなく、登録と同時に認証してログインする処理も行うことにします。まずテンプレートを用意しましょう。

template/registration/create.html

```
{% extends 'base/base.html' %}
{% block title %}ITについて切磋琢磨する掲示板 - {{ block.super }}{% endblock %}
{% block content %}
<div class="ui grid stackable">
    <div class="eleven wide column">
        <div class="ui breadcrumb">
            <a href="{% url 'base:top' %}" class="section">TOP</a>
            <i class="right angle icon divider"></i>
            <a class="active section">ユーザー作成</a>
        </div>
        <div class="ui segment">
            <div class="content">
                <div class="header"><h3>ユーザー作成</h3></div>
                <form class="ui form" action="" method="POST">
                    {% csrf_token %}
                    {{form.as_p}}
                    <button class="ui orange button" name="next" value="confirm" type="sub
                </form>
            </div>
        </div>
    </div>
    {% include 'base/sidebar.html' %}
</div>
{% endblock %}
```

template/registration/create_confirm.html

```
{% extends 'base/base.html' %}
{% block title %}ITについて切磋琢磨する掲示板 - {{ block.super }}{% endblock %}
{% block content %}
<div class="ui grid stackable">
    <div class="eleven wide column">
        <div class="ui breadcrumb">
            <a href="{% url 'base:top' %}" class="section">TOP</a>
            <i class="right angle icon divider"></i>
            <a class="active section">ユーザー作成</a>
        </div>
        <div class="ui segment">
            <div class="content">
                <div class="header"><h3>ユーザー作成</h3></div>
                <table  class="ui celled table table table-hover" >
                    <tr><td>お名前</td><td>{{form.cleaned_data.username}}</td></tr>
                    <tr><td>パスワード</td><td>********</td></tr>
                </table>
                <form class="ui form" action="" method="POST">
                    {% csrf_token %}
                    {% for field in form %}
                    {{field.as_hidden}}
                    {% endfor %}
                    <button class="ui button" name="next" value="back" type="submit">修正<
                    <button class="ui orange button" name="next" value="regist" type="subm
                </form>
            </div>
        </div>
    </div>
    {% include 'base/sidebar.html' %}
</div>
{% endblock %}
```

ここまでは特に問題ないと思います。ではビューを作っていきましょう。

accounts/views.py

```
from django.shortcuts import render, redirect
from django.views.generic import TemplateView, FormView
from django.contrib.auth.forms import UserCreationForm
from django.contrib.auth.models import User
from django.contrib.auth.mixins import LoginRequiredMixin
from django.urls import reverse_lazy
from django.contrib.auth import login, authenticate

# Create your views here.

class UserCreateView(FormView):
    form_class = UserCreationForm
    template_name = 'registration/create.html'
    success_url = reverse_lazy('base:top')
    def form_valid(self, form):
        print(self.request.POST['next'])
        if self.request.POST['next'] == 'back':
            return render(self.request, 'registration/create.html', {'form': form})
        elif self.request.POST['next'] == 'confirm':
            return render(self.request, 'registration/create_confirm.html', {'form': form}
        elif self.request.POST['next'] == 'regist':
            form.save()
            # 認証
            user = authenticate(
                username=form.cleaned_data['username'],
                password=form.cleaned_data['password1'],
            )
            # ログイン
            login(self.request, user)
            return super().form_valid(form)
        else:
            # 通常このルートは通らない
            return redirect(reverse_lazy('base:top'))
```

簡単に解説しますと、POSTで受ける'next'パラメータの値によって挙動がコントロールされているのは問題ないと思います。この仕組みはトピック登録画面と同様です。問題はform_classで設定されているUserCreationFormですね。これはDjangoで用意されたフォームでユーザー登録用のフォームです。今回はこのフォームクラスをそのまま使いました。もし、カスタマイズが必要な場合はUserCreationFormを継承したクラスを作成すれば良いということになります。次に、登録処理のところですね。form.save()メソッドを呼んでユーザー情報を登録してUser情報を登録しています。次にauthenticate関数を呼んで認証処理を行っています。authenticate関数が呼ばれると認証用のバックエンドのリストが呼び出されて順番に認証バックエンドで認証できるかが試されていきます。無事に認証が通れば認証バックエンドと紐づけされたユーザーが返される仕組みです。このユーザーをlogin関数で処理することでセッションにユーザーデータをもたせてログイン処理を行っています。

ではこのビューにアクセスするURLを作成しましょう。
accounts/urls.py

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

app_name = 'accounts'

urlpatterns = [
    path('create/', views.UserCreateView.as_view(), name="create"),
]
```

早速確認したいところですが、ここでユーザー登録してログインした際にヘッダー部分の表示が変わるように変更しておきましょう。templates/base/base.htmlを修正します。

```
  {% load static %}
  <!DOCTYPE html>
  <head>
      <meta charset="UTF-8">
      <meta http-equiv="content-language" content="ja">
      <meta name="viewport" content="width=device-width, initial-scale=1.0, maximum-scale=
      {% block meta_tag %}{% endblock %}
      <link href="{% static 'css/semantic.css' %}" rel="stylesheet">
      {% block css %}{% endblock %}
      <title>
          {% block title %}IT学習ちゃんねる{% endblock %}
      </title>
  </head>
  <body>
      <div class="ui stackable inverted menu">
          <a href="{% url 'base:top' %}" class="header item">
              IT学習ちゃんねる
          </a>
          <a href="{% url 'base:about' %}" class="item">
              このサイトはなに？
          </a>
          <a class="item" href="{% url 'thread:create_topic' %}">
              トピック作成
          </a>
-         <div class="right menu">
-             <a class="item" href="">ログイン</a>
-             <a class="item" href="">ユーザー登録</a>
-         </div>
+         <div class="right menu">
+             {% if user.is_authenticated %}
+             <a class="item" href="">ユーザー情報</a>
+             <a class="item" href="">ログアウト</a>
+             {% else %}
+             <a class="item" href="">ログイン</a>
+             <a class="item" href="{% url 'accounts:create' %}">ユーザー登録</a>
+             {% endif %}
+         </div>
      </div>

      <div class="ui container" style="min-height:100vh;">
          {% block content %}
          {% endblock %}
      </div>
      <div class="ui inverted stackable footer segment">
          <div class="ui container center aligned">
              <div class="ui horizontal inverted small divided link list">
                  <a href="{% url 'base:top' %}" class="item">© 2019 IT学習ちゃんねる(仮)</
                  <a href="{% url 'base:terms' %}" class="item">利用規約</a>
```

```
                <a href="{% url 'base:policy' %}" class="item">プライバシーポリシー</a>
            </div>
        </div>
    </div>
    <script src="https://code.jquery.com/jquery-3.1.1.js"></script>
    <script type="text/javascript" src="{% static 'js/semantic.js' %}"></script>
    {% block js %}{% endblock %}
</body>
```

ポイントは条件分岐で使用されているuser.is_authenticated関数ですねこれはユーザーが認証されているかを判定する関数です。ユーザーがログインしてに称された状態であれば「ユーザー情報」と「ログアウト」を表示するようにしています。ユーザー登録だけは画面が作成されていますのでURLを設定します。

それでは確認してみましょう。ユーザーを登録してTOPページに遷移してヘッダーの表示が変化すればOKです。
まずはヘッダーの「ユーザー登録」からアクセス

ユーザーを作成

TOP › ユーザー作成

ユーザー作成

ユーザー名:

user1

この項目は必須です。半角アルファベット、半角数字、@/./+/-/_ で150文字以下にしてください。

パスワード:

••••••••••

- あなたの他の個人情報と似ているパスワードにはできません。
- パスワードは最低８文字以上必要です。
- よく使われるパスワードにはできません。
- 数字だけのパスワードにはできません。

パスワードの確認:

••••••••••

確認のため、再度パスワードを入力してください。

確認

TOP › ユーザー作成

ユーザー作成

お名前	user1
パスワード	********

修正 登録

完了するとヘッダーの表示が変化する。（ログイン済み）

最後に

今回は会員サイトを作る上で必須となる会員登録ページの作り方でした。UserCreationFormを使うことで余分な手間なく出来たと思います。次回は登録したユーザーの情報を表示するページを作っていきます。

2-12. タイムゾーンと日時オブジェクトを扱う

2019年4月18日　コメントをする

今回のテーマは「タイムゾーンと日時オブジェクトを扱う」です。いよいよ第二章も最終回です。もうDjnagoの特徴や使い方が大分見えてきましたよね。今回はデータベースに保存されている日時を利用して現在時刻との差分を計算していきます。今回扱う範囲は公式ドキュメントのタイムゾーンに詳細説明があります。

日時オブジェクトのnativeとawareについて

settings.pyにてUSE_TZ = Trueにしている場合タイムゾーンがサポートされます。この状態ではDjangoはタイムゾーンを認識する日時オブジェクト(awareな日時オブジェクト)を使用します。本サイトのようにstartprojectコマンドによってプロジェクトを生成した場合は初期設定としてUSE_TZ=Trueの設定になっています。よって現状ではawareな日時オブジェクトを使用してきました。タイムゾーンが有効となっている場合、データベースにはUTCで日時を保存しています。Djangoはテンプレートやフォーム等で表示する際に、設定されたタイムゾーンで変換をしています。

現在時刻の取得に関して

pythonで現在時刻を取得する際にdatetime.datetime.now()と覚えている方も多いのではないでしょうか？この方法ではnativeの日時オブジェクトが取得されるため、awareな日時オブジェクトとの比較は出来ません。データベースから取得した日時オブジェクトと現在時刻を比較するにはどうしたら良いのでしょうか？Djangoにはdjango.utils.timezoneモジュールがあります。このモジュールのnow関数を使うことで現時刻の日時オブジェクトを適当なモードで取得することが出来ます。now関数の中身を見てみましょう。

django/utils/timezone.py

```
def now():
    """
    Return an aware or naive datetime.datetime, depending on settings.USE_TZ.
    """
    if settings.USE_TZ:
        # timeit shows that datetime.now(tz=utc) is 24% slower
        return datetime.utcnow().replace(tzinfo=utc)
    else:
        return datetime.now()
```

このようにUSE_TZによって返すオブジェクトを変えています。タイムゾーンが無効(USE_TZ=False)の場合にはnativeの日時オブジェクトが返されるのが理解できると思います。

よってDjango内で現在時刻を取得する際にはtimezoneモジュールをインポートしてnow関数を呼び出すのが良いでしょう。

データベース

簡単にデータベースについて触れておきたいと思います。筆者としてはタイムゾーンは常に有効にして使用した方が良いと考えていますが、タイムゾーンの有効・無効を切り替える場合もあると思います。PostgreSQLはタイムゾーン情報をデータベースに保存しているために、タイムゾーンの有効・無効は自由に切り替えられます。しかしそれ以外のデータベースに関してはタイムゾーンがを無効に切り替えた場合にはUTCからネイティブなdatetimeに変換する必要があります。（参考：公式ドキュメント）

現在時刻との差分を計算してNEWラベルをつける

さて、今回は演習として一時間以内に新しく作成されたトピックはトップページにNEWラベルを表示するようにしましょう。トップページを表示しているビューはbase/views.pyのTopicListViewクラスですので、ここに手を加えていきます。

base/views.py(一部抜粋)

```
from django.utils import timezone

class TopicListView(ListView):
    template_name = 'base/top.html'
    # model = Topic
    queryset = Topic.objects.order_by('-created')
    context_object_name = 'topic_list'

    def __init__(self, *args, **kwargs):
        super().__init__(*args, **kwargs)
        self.new_list = []

    def get_queryset(self):
        topic_list = Topic.objects.order_by('-created')
        self.new_list = self._make_new_list(topic_list)
        return topic_list

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

    def _make_new_list(self, topic_list):
        def pickup_topic(topic):
            now = timezone.now()
            diff = (now - topic.created).total_seconds() / (60 * 60)
            if diff > 1:
                return False
            else:
                return True
        return list(map(lambda x : x.id ,filter(pickup_topic, topic_list)))
```

解説は不要かと思いますが、現在時刻をtimezone.now()で取得してデータベースに格納されているtopic.createdと差を求めて秒を時間に変換しています。この時間が 1 時間以下のトピックのIDのみのリストを作成して返していますね。コンテキストにnew_listを渡すためnew_listをインスタンス変数として渡しています。

ラベルを表示するようにテンプレートも変更しましょう。

templates/base/top.html(一部抜粋)

```
{% extends 'base/base.html' %}
{% block title %}ITについて切磋琢磨する掲示板 - {{ block.super }}{% endblock %}
{% block content %}
<div class="ui grid stackable">
    <div class="eleven wide column">
        <div class="ui breadcrumb">
            <a class="active section">TOP</a>
        </div>
        <div class="ui segment">
            <div class="content">
                <div class="header"><h3>新着スレッド</h3></div>
                <div class="ui divided items">
                    {% for topic in topic_list %}
                    <div class="item">
                        <div class="content">
                            <div class="header">
                                <a href="{% url 'thread:topic' pk=topic.id %}">
                                    <h4>
                                        {% if topic.id in new_list %}
                                        <div class="ui violet horizontal label">new</div>
                                        {% endif %}
                                        {{topic.title}}
                                    </h4>
                                </a>
                            </div>
                            <div class="meta">
                                <span class="name">{{topic.user_name}}</span>
                                <span class="date">{{topic.created}}</span>
                            </div>
                        </div>
                    </div>
                    {% endfor %}
                </div>
            </div>
        </div>
    </div>
    {% include 'base/sidebar.html' %}
</div>
{% endblock %}
```

全て掲載しましたが、修正点はトピックタイトルの部分のみです。

では確認してみましょう。新規にトピックを作成するとNEWラベルがつきますね。

最後に

ここまで、Djangoの機能をつまみ食いしながら紹介してきました。少々無理のある機能もあり、掲示板というお題で始めてしまってよかったのか悩む場面もありましたが、何とかそれなりの機能を有した掲示板になってきたのではないでしょうか？次章からはDjangoの認証機能を利用して活きます。ここまで作成した掲示板を会員サイトに修正していく予定です。好ご期待！