3-12. ログイン状態に応じて動的に入力フォームを変化させる

2019年5月22日　コメントをする

今回のテーマは「ログイン状態に応じて動的に入力フォームを変化させる」です。せっかくログイン機能を付与したので、ログインユーザーは名前やメールアドレスを入力しなくても投稿出来るように、しましょう。また、未ログインユーザーはゲストとしてメールアドレスと名前を入力すればスレッド作成やコメント投稿が出来るように修正していきましょう。

※本ページは「3-11. 認証バックエンドをカスタマイズしてログイン方法を変更する」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。

モデルの修正

ログインユーザーとTopicモデル、Commentモデルを結びつけましょう。また、ゲストユーザー用にメールアドレスも追加しましょう。

thread/forms.py(一部抜粋)

```
class TopicModelForm(forms.ModelForm):
    # ゲストユーザー用入力項目
    user_name = forms.CharField(
        label='ゲストユーザー名',
        required=True,
    )
    email = forms.EmailField(
        label='メールアドレス',
        required=True,
    )

    class Meta:
        model=Topic
        fields=[
            'title',
            # 'user_name',
            # 'email',
            'category',
            'message',
        ]
        widgets = {
            'title' : forms.TextInput(attrs={'class': 'hoge'}),
            # 'user_name' : forms.TextInput(attrs={'value': '名無し'}),
        }

    def __init__(self, *args, **kwargs):
        kwargs.setdefault('label_suffix', '')
        super().__init__(*args, **kwargs)
        self.fields['category'].empty_label = '選択して下さい'
        self.fields['user_name'].widget.attrs['value'] = 'anonymous'

    def save(self, commit=True):
        topic = super().save(commit=False)
        topic.user_name = self.cleaned_data['user_name']
        topic.email = self.cleaned_data['email']
        if commit:
            topic.save()
        return topic

        # self.fields['user_name'].widget.attrs['value'] = '匿名'
```

ゲストユーザー用のスレッド作成フォームの作成

ゲストユーザーがスレッドを作成出来るようにTopicModelFormを改修しましょう。thread/forms.pyを以下のように修正します。

thread/forms.py(一部抜粋)

```
class TopicModelForm(forms.ModelForm):
    # ゲストユーザー用入力項目
    user_name = forms.CharField(
        label='ゲストユーザー名',
        required=True,
    )
    email = forms.EmailField(
        label='メールアドレス',
        required=True,
    )

    class Meta:
        model=Topic
        fields=[
            'title',
            # 'user_name',
            # 'email',
            'category',
            'message',
        ]
        widgets = {
            'title' : forms.TextInput(attrs={'class': 'hoge'}),
            # 'user_name' : forms.TextInput(attrs={'value': '名無し'}),
        }

    def __init__(self, *args, **kwargs):
        kwargs.setdefault('label_suffix', '')
        super().__init__(*args, **kwargs)
        self.fields['category'].empty_label = '選択して下さい'
        self.fields['user_name'].widget.attrs['value'] = 'anonymous'

    def save(self, commit=True):
        topic = super().save(commit=False)
        topic.user_name = self.cleaned_data['user_name']
        topic.email = self.cleaned_data['email']
        if commit:
            topic.save()
        return topic

        # self.fields['user_name'].widget.attrs['value'] = '匿名'
```

敢えて、user_nameとemailをクラス変数として追加しています。これはuser_nameとemailについては管理画面では任意項目として、ゲストユーザーの入力フォームのみで必須項目としたいためです。管理画面においても必須項目とする場合にはMetaクラス内のfieldsで指定しても構わないと思います。

ログインユーザー用のスレッド作成フォームの作成

次にログインユーザーが入力するフォームを作成していきます。thread/forms.pyに以下のように追記します。

thread/forms.py(一部抜粋)

```
class LoginedUserTopicModelForm(forms.ModelForm):
    class Meta:
        model=Topic
        fields=[
            'title',
            'category',
            'message',
        ]
        widgets = {
            'title' : forms.TextInput(attrs={'class': 'hoge'}),
        }

    def __init__(self, *args, **kwargs):
        kwargs.setdefault('label_suffix', '')
        super().__init__(*args, **kwargs)
        self.fields['category'].empty_label = '選択して下さい'
        # self.fields['user_name'].widget.attrs['value'] = '匿名'

    def save(self, user, commit=True, **kwargs):
        topic = super().save(commit=False)
        topic.user = user
        if commit:
            topic.save()
        return topic
```

特に解説は不要かと思います。save関数はオーバーライドしてカスタムユーザーを指定して保存するようにしています。

ゲストユーザー用のコメント投稿フォームの作成

次にゲストユーザーがコメント投稿するためのフォームを用意するためにCommentModelFormを改修します。thread/forms.pyを修正します。

thread/forms.py(一部抜粋)

```
class CommentModelForm(forms.ModelForm):
    # ゲストユーザー用入力項目
    user_name = forms.CharField(
        label='お名前',
        required=True,
    )
    email = forms.EmailField(
        label='メールアドレス',
        required=True,
    )

    class Meta:
        model = Comment
        fields = [
            'message',
            'image',
        ]

    def __init__(self, *args, **kwargs):
        kwargs.setdefault('label_suffix', '')
        super().__init__(*args, **kwargs)
        self.fields['user_name'].widget.attrs['value'] = 'anonymous'

    def save_with_topic(self, topic_id, commit=True, **kwargs):
        comment = self.save(commit=False)
        comment.topic = Topic.objects.get(id=topic_id)
        comment.no = Comment.objects.filter(topic_id=topic_id).count() + 1
        comment.user_name = self.cleaned_data['user_name']
        comment.email = self.cleaned_data['email']
        if commit:
            comment.save()
        return comment
```

改修方針はTopicModelFormと同じですね。save_with_topic関数でもuser_nameとemailをUserオブジェクトに渡していますので忘れないようにして下さい。

ログインユーザー用のコメント作成フォームの作成

フォームの準備の最後はログインユーザーがコメント投稿するためのフォームの準備です。
thread/forms.pyに追記していきます。

thread/forms.py(一部抜粋)

```
class LoginedUserCommentModelForm(forms.ModelForm):
    class Meta:
        model = Comment
        fields = [
            'message',
            'image',
        ]

    def __init__(self, *args, **kwargs):
        kwargs.setdefault('label_suffix', '')
        super().__init__(*args, **kwargs)

    def save_with_topic(self, topic_id, user, commit=True, **kwargs):
        comment = self.save(commit=False)
        comment.topic = Topic.objects.get(id=topic_id)
        comment.no = Comment.objects.filter(topic_id=topic_id).count() + 1
        comment.user = user
        if commit:
            comment.save()
        return comment
```

特に難しい部分はないと思います。save_with_topic関数には仮引数にuserがあるので注意して下さい。

ログイン状態に応じてフォームを変更するようにビューを修正する

さて、ここまで準備したフォームをログイン状態によって動的に使い分けていきます。ゲストユーザーの状態(未ログイン状態)ではCommentModelFormとTopicModelFormを使用します。そしてログイン状態ではLoginedUserCommentModelFormとLoginedUserTopicModelFormを使用します。この切り替えを動的に行うためにビューの中でget_form_class関数をオーバライドします。ではthread/views.pyの改修を見ていきましょう。

thread/views.py(一部抜粋)

```
#importはファイル文頭で
from . forms import (
    TopicModelForm, TopicForm, CommentModelForm,
    LoginedUserTopicModelForm, LoginedUserCommentModelForm
)
from . models import Topic, Category, Comment


class TopicCreateView(CreateView):
    template_name = 'thread/create_topic.html'
    form_class = LoginedUserTopicModelForm
    model = Topic
    success_url = reverse_lazy('base:top')

    def get_form_class(self):
        '''
        ログイン状態によってフォームを動的に変更する
        '''
        if self.request.user.is_authenticated:
            return LoginedUserTopicModelForm
        else:
            return TopicModelForm

    def form_valid(self, form):
        ctx = {'form': form}
        if self.request.POST.get('next', '') == 'confirm':
            ctx['category'] = form.cleaned_data['category']
            return render(self.request, 'thread/confirm_topic.html', ctx)
        elif self.request.POST.get('next', '') == 'back':
            return render(self.request, 'thread/create_topic.html', ctx)
        elif self.request.POST.get('next', '') == 'create':
            form.save(self.request.user)
            # メール送信処理
            template = get_template('thread/mail/topic_mail.html')
            user_name = self.request.user.username if self.request.user else form.cleaned_
            mail_ctx={
                'title': form.cleaned_data['title'],
                'user_name': user_name,
                'message': form.cleaned_data['message'],
            }
            EmailMessage(
                subject='トピック作成: ' + form.cleaned_data['title'],
                body=template.render(mail_ctx),
                from_email='hogehoge@example.com',
                to=['admin@example.com'],
                cc=['admin2@example.com'],
                bcc=['admin3@example.com'],
            ).send()
            # return super().form_valid(form)
```

```
            return redirect(self.success_url)
        else:
            # 正常動作ではここは通らない。エラーページへの遷移でも良い
            return redirect(reverse_lazy('base:top'))

class TopicAndCommentView(FormView):
    template_name = 'thread/detail_topic.html'
    form_class = LoginedUserCommentModelForm

    def get_form_class(self):
        '''
        ログイン状態によってフォームを動的に変更する
        '''
        if self.request.user.is_authenticated:
            return LoginedUserCommentModelForm
        else:
            return CommentModelForm

    def form_valid(self, form):
        # comment = form.save(commit=False)
        # comment.topic = Topic.objects.get(id=self.kwargs['pk'])
        # comment.no = Comment.objects.filter(topic=self.kwargs['pk']).count() + 1
        # comment.save()

        # Comment.objects.create_comment(
        #     user_name=form.cleaned_data['user_name'],
        #     message=form.cleaned_data['message'],
        #     topic_id=self.kwargs['pk'],
        #     image=form.cleaned_data['image']
        # )
        kwargs = {}
        if self.request.user.is_authenticated:
            kwargs['user'] = self.request.user

        form.save_with_topic(self.kwargs.get('pk'), **kwargs)
        response = super().form_valid(form)
        return response

    def get_success_url(self):
        return reverse_lazy('thread:topic', kwargs={'pk': self.kwargs['pk']})

    def get_context_data(self):
        ctx = super().get_context_data()
        ctx['topic'] = Topic.objects.get(id=self.kwargs['pk'])
        ctx['comment_list'] = Comment.objects.filter(
                topic_id=self.kwargs['pk']).annotate(vote_count=Count('vote')).order_by('n
        return ctx
```

ポイントはrequest.user.is_authenticated()メソッドによって使用するフォームを使い分けているところですね。TopicオブジェクトやCommentオブジェクトの保存処理は各フォームオブジェクトの同一メソッドを呼び出すことで共通化しています。

念のため、上記のビュークラスを使用するようにURLも確認しておきましょう。

thread/urls.py

```
from django.urls import path

from . import views
app_name = 'thread'


urlpatterns = [
    # path('create_topic/', views.TocicCreateViewBySession.as_view(), name='create_topic')
    path('create_topic/', views.TopicCreateView.as_view(), name='create_topic'),
    # path('create_topic/', views.topic_create, name='create_topic'),
    path('/', views.TopicAndCommentView.as_view(), name='topic'),
    # path('category//', views.CategoryView.as_view(), name='category'),
    path('category//', views.show_catgegory, name='category'),
]
```

テンプレートの修正

ログインしている場合とゲストユーザーの場合で表示を切り替えるため、テンプレートにも修正を加えましょう。テンプレートにはuserが渡されていますので、このuserで判断をしていきます。

スレッド作成画面
templates/thread/create_topic.html.py

```
{% extends 'base/base.html' %}
{% block title %}スレッド作成 - {{ 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">スレッド作成</a>
        </div>
        <div class="ui segment">
            <div class="content">
                <div class="header"><h3>スレッド作成</h3></div>
                {% if not user.is_authenticated %}
                <div class="ui warning message">
                    <p>ゲストユーザーでスレッド作成しますか？ログインする場合は以下のボタン
                    <a class="ui button orange" href="/accounts/login?next=/thread/create_
                </div>
                {% endif %}
                <form class="ui form" action="{% url 'thread:create_topic' %}" method="POS
                    {% csrf_token %}
                    <!-- {{form.as_p}} -->
                    {% for field in form %}
                    <div class="field">{{field.label_tag}}{{field}}</div>
                        {% for error in field.errors%}
                        <p style="color: red;">{{error}}</p>
                        {% endfor%}
                    {% endfor %}
                    <button type="submit" class="ui button" name="next" value="confirm">作
                </form>
            </div>
        </div>
    </div>
    {% include 'base/sidebar.html' %}
</div>
{% endblock %}
```

templates/thread/confirm_topic.html.py

```
{% extends 'base/base.html' %}
{% block title %}スレッド作成 - {{ 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">スレッド作成</a>
        </div>
        <div class="ui segment">
            <div class="content">
                <div class="header"><h3>スレッド作成</h3></div>
                <p>内容を確認してください</p>
                <table  class="ui celled table table table-hover" >
                    <tr><td>タイトル</td><td>{{form.title.value}}</td></tr>
                    {% if user.is_authenticated %}
                    <tr><td>お名前</td><td>{{user.username}}</td></tr>
                    {% else %}
                    <tr><td>ゲスト名</td><td>{{form.cleaned_data.user_name}}</td></tr>
                    {% endif %}
                    <tr><td>カテゴリー</td><td>{{form.cleaned_data.category}}</td></tr>
                    <tr><td>本文</td><td><pre>{{form.message.value}}</pre></td></tr>
                </table>
                <form class="ui form" action="{% url 'thread:create_topic' %}" method="POS
                    {% csrf_token %}
                    {% for field in form %}
                        {{field.as_hidden}}
                    {% endfor %}
                    <button class="ui button grey" type="submit" name="next" value="back">
                    <button class="ui button orange" type="submit" name="next" value="crea
                </form>
            </div>
        </div>
    </div>
    {% include 'base/sidebar.html' %}
</div>
{% endblock %}
```

templates/thread/detail_topic.html

```
{% extends 'base/base.html' %}
{% block title %}トピック作成 - {{ block.super }}{% endblock %}
{% block content %}
{% load threadfilters %}
{% load static %}
<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 'thread:category' url_code=topic.category.url_code %}" class="
            <i class="right angle icon divider"></i>
            <a class="active section">{{topic.title}}</a>
        </div>
        <div class="ui segment">
            <div class="content">
                <div class="header"><h3>{{topic.title}}</h3></div>
                <p>{% if topic.user %}{{topic.user}}{% else %}{{topic.user_name}}(Guest){%
                <div class="ui segment">
                    <p><pre>{{topic.message}}</pre></p>
                </div>
            </div>
        </div>
        <!--コメント表示-->
        <div class="ui segment">
            {% if comment_list %}
            {% for comment in comment_list %}
            <div class="ui segment secondary">
                <p>{{comment.no}}. {% if comment.user %}{{comment.user.username}}{% else %
                <br>{{comment.created}}</p>
                {% if comment.pub_flg %}
                <p>{{comment.message | comment_filter | safe}}</p>
                    {% if comment.image %}
                    <a href="{{comment.image.url}}" target="_blank" rel="noopener noreferr
                    {% endif %}
                <div class="ui right aligned vertical segment">
                    <div class="vote_button" style="cursor: pointer;"
                        data-comment-id="{{comment.id}}" data-count="{{comment.vote_count}
                        <i class="heart outline icon"></i>
                        <span class="vote_counter">
                            {% if comment.vote_count > 0 %}{{comment.vote_count}}{% endif
                        </span>
                    </div>
                </div>
                {% else %}
                <p style="color: grey">コメントは非表示とされました</p>
                {% endif %}
            </div>
```

```
            {% endfor %}
            {% else %}
            <div class="ui warning message"><p>まだコメントはありません</p></div>
            {% endif %}
        </div>
        <!--//コメント表示-->
        <!--コメント投稿-->
        <h4>コメント投稿</h4>
        {% if not user.is_authenticated %}
        <div class="ui warning message">
            <p>ゲストユーザーで投稿しますか？ログインする場合は以下のボタンでログインして下さ
            <a class="ui button orange" href="/accounts/login?next=/thread/{{topic.id}}/">
        </div>
        {% endif %}
        <div class="ui segment">
            <form class="ui form" action="" method="POST" enctype="multipart/form-data">
                {% csrf_token %}
                {{form.as_p}}
                <button class="ui button orange" type="submit">コメント投稿</button>
            </form>
        </div>
        <!--//コメント投稿-->
    </div>
    {% include 'base/sidebar.html' %}
</div>
{% endblock %}
{% block js %}
<script src="{% static 'js/vote.js' %}" type='text/javascript'></script>
{% endblock %}
```

基本的にはuser.is_authenticatedで場合分けしていくことでログインしている場合としていない場合を区別することが出来ます。

動作確認

では、動作確認をしてみましょう。

まずはゲストユーザーでスレッド作成をしてみましょう。

TOP › スレッド作成

スレッド作成

内容を確認してください

タイトル	Djangoの学習サイト
ゲスト名	anonymous
カテゴリー	プログラミング
本文	Djangoを学習するのに良いサイトないですか？

戻る　作成

ゲストユーザーでのコメント作成も出来ますね。

コメント投稿

ゲストユーザーで投稿しますか？ログインする場合は以下のボタンでログインして下さい。

ログイン

投稿内容

Django学習帳が良いと思います。

投稿画像

ファイルを選択　選択されていません

お名前

anonymous

メールアドレス

hoge@example.com

コメント投稿

次にログインユーザーでスレッド作成してみましょう。

ログインユーザーでのコメント投稿画面です。

最後に

3-11. 認証バックエンドをカスタマイズしてログイン方法を変更する

2019年5月20日　コメントをする

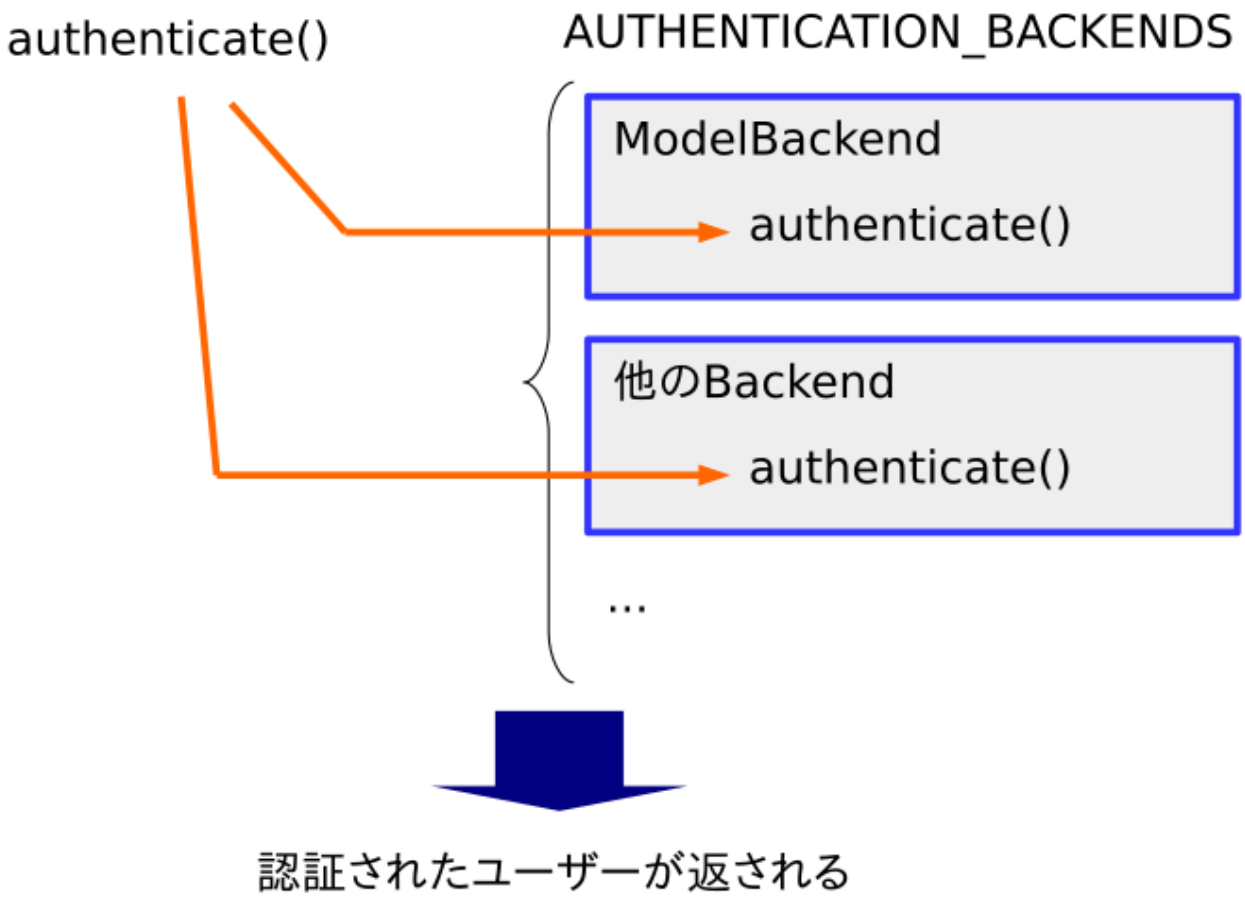
今回のテーマは「3-11. 認証バックエンドをカスタマイズしてログイン方法を変更する」です。これまで、Django標準のログイン方法としてユーザー名とパスワードでログインするログイン画面でした。しかしメールアドレスでログインする方法に変更したい場合もあると思います。今回はDjangoのユーザー認証について見ていきましょう。

※本ページは「3-10. 独自カスタマイズのユーザーを使用する」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のログインの仕組み

djangoの認証は認証用のバックエンドによって行われます。このバックエンドはsettings.pyのAUTHENTICATION_BACKENDSに設定します。AUTHENTICATION_BACKENDSのデフォルト値は'django.contrib.auth.backends.ModelBackend'１つですが、バックエンドは配列として追加することが出来ます。

Djangoの認証はdjango.contrib.auth.authenticate関数で行われます。このauthenticate関数が呼ばれるとAUTHENTICATION_BACKENDSに記載されたバックエンドのauthenticate関数を順番に呼び出します。認証に失敗すれば次のバックエンドを呼び出します。もし認証に成功すれば、認証されたユーザーを返す仕組みです。



今回は独自に実装したフォームの中のclean関数をオーバーライドしてauthenticate関数を呼び出し認証処理を行います。もし認証に失敗した場合はバリデーションエラーを返す仕組みです。

認証バックエンドをカスタマイズする

デフォルトのModelBackendに加えてカスタマイズしたバックエンドを追加しましょう。
accounts/backends.pyを新規作成します。ファイル名はこの名前でなくても大丈夫です。

accounts/backends.py

```
from django.contrib.auth.backends import ModelBackend
from .models import User

class EmailAuthBackend(ModelBackend):
    def authenticate(self, request, email=None, password=None, **kwargs):
        try:
            user = User.objects.get(email=email)
        except User.DoseNotExist:
            return None
        else:
            if user.check_password(password) and self.user_can_authenticate(user):
                return user
```

作成したバックエンドを設定に追加する
作成したバックエンドを追加するためにはmysite/settings.pyに追加します。インポートが多いのでEmailAuthenticationForm以外のクラスも表示していますが、追加するのはEmailAuthenticationFormのみです。このEmailAuthenticationFormはdjango.contrib.auth.form.AuthenticationFormをベースとしてメールアドレス用に修正しています。

mysite/settings.py

```
+ AUTHENTICATION_BACKENDS = [
+     'django.contrib.auth.backends.ModelBackend',
+     'accounts.backends.EmailAuthBackend',
+ ]
```

メールアドレスでのログイン用フォームを用意する

メールアドレスで認証するためのフォームを作成していきます。

accounts/forms.py

```python
from django.contrib.auth import (
    authenticate, get_user_model
)
from django.forms import ModelForm, Form
from django.forms.fields import EmailField
from django import forms

# from django.contrib.auth.models import User
# from django.contrib.auth import get_user_model
from django.contrib.auth.forms import (
    AuthenticationForm, PasswordChangeForm,
    PasswordResetForm, SetPasswordForm,
    UserChangeForm, UserCreationForm
)
from django.utils.translation import gettext_lazy as _
from django.utils.text import capfirst

from .models import User

UserModel = get_user_model()

class UserInfoChangeForm(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()

class EmailChangeForm(ModelForm):
    class Meta:
```

```
        model = User
        fields = ['email']

    def __init__(self, email=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

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


class CustomAuthenticationForm(AuthenticationForm):
    def __init__(self, *args, **kwargs):
        kwargs.setdefault('label_suffix', '')
        super().__init__(*args, **kwargs)

class CustomPasswordChangeForm(PasswordChangeForm):
    def __init__(self, *args, **kwargs):
        kwargs.setdefault('label_suffix', '')
        super().__init__(*args, **kwargs)

class CustomPasswordResetForm(PasswordResetForm):
    def __init__(self, *args, **kwargs):
        kwargs.setdefault('label_suffix', '')
        super().__init__(*args, **kwargs)

class CustomSetPasswordForm(SetPasswordForm):
    def __init__(self, *args, **kwargs):
        kwargs.setdefault('label_suffix', '')
        super().__init__(*args, **kwargs)

class AdminUserCreationForm(UserCreationForm):
    class Meta:
        model = User
        fields = ('username', 'email')

    # def save(self, commit=True):
    #     user = User.objects.create_user(
    #         self.cleaned_data["name"],
    #         self.cleaned_data["email"],
    #         self.cleaned_data["password1"],
    #     )
    #     return user

class CustomUserChangeForm(UserChangeForm):
    class Meta:
        model = User
```

```
        fields = '__all__'

class CustomUserCreationForm(UserCreationForm):
    class Meta:
        model = User
        fields = ('username', 'email')

    def __init__(self, *args, **kwargs):
        kwargs.setdefault('label_suffix','')
        super().__init__(*args, **kwargs)

class EmailAuthenticationForm(Form):
    """
    django.contrib.auth.form.AuthenticationFormをベースに改変
    """
    email = EmailField(
        label=_('Email'),
        widget=forms.EmailInput(attrs={'autofocus': True,})
    )
    password = forms.CharField(
        label=_("Password"),
        strip=False,
        widget=forms.PasswordInput,
    )

    error_messages = {
        'invalid_login': _(
            "Please enter a correct %(username)s and password. Note that both "
            "fields may be case-sensitive."
        ),
        'inactive': _("This account is inactive."),
    }

    def __init__(self, request=None, *args, **kwargs):
        """
        The 'request' parameter is set for custom auth use by subclasses.
        The form data comes in via the standard 'data' kwarg.
        """
        self.request = request
        self.user_cache = None
        kwargs.setdefault('label_suffix', '')
        super().__init__(*args, **kwargs)

        # Set the max length and label for the "username" field.
        self.email_field = UserModel._meta.get_field(UserModel.USERNAME_FIELD)
        self.fields['email'].max_length = self.email_field.max_length or 254
        if self.fields['email'].label is None:
            self.fields['email'].label = capfirst(self.email_field.verbose_name)
        for field in self.fields.values():
            field.widget.attrs["class"] = "form-control"
            field.widget.attrs["placeholder"] = field.label
```

```
    def clean(self):
        email = self.cleaned_data.get('email')
        password = self.cleaned_data.get('password')

        if email is not None and password:
            self.user_cache = authenticate(self.request, email=email, password=password)
            if self.user_cache is None:
                raise self.get_invalid_login_error()
            else:
                self.confirm_login_allowed(self.user_cache)

        return self.cleaned_data

    def confirm_login_allowed(self, user):
        """
        Controls whether the given User may log in. This is a policy setting,
        independent of end-user authentication. This default behavior is to
        allow login by active users, and reject login by inactive users.

        If the given user cannot log in, this method should raise a
        ``forms.ValidationError``.

        If the given user may log in, this method should return None.
        """
        if not user.is_active:
            raise forms.ValidationError(
                self.error_messages['inactive'],
                code='inactive',
            )

    def get_user(self):
        return self.user_cache

    def get_invalid_login_error(self):
        return forms.ValidationError(
            self.error_messages['invalid_login'],
            code='invalid_login',
            params={'username': _('Email')},
        )
```

最初の解説でも触れましたが、clean関数内でauthenticate関数を呼び認証処理を行っています。こうすることで各バックエンドを呼び出して認証することが出来ます。

ビューの変更

作成したEmailAuthenticationFormを使用するようにビューを変更しましょう。
accounts/views.py(一部抜粋)

```
  from .forms import (
      UserInfoChangeForm,
      CustomAuthenticationForm, CustomPasswordChangeForm,
      CustomPasswordResetForm, CustomSetPasswordForm,
      CustomUserChangeForm, CustomUserCreationForm, EmailChangeForm,
+     EmailAuthenticationForm
  )

#一部省略・・・

  class CustomLoginView(LoginView):
-     form_class = CustomAuthenticationForm
+     # form_class = CustomAuthenticationForm
+     form_class = EmailAuthenticationForm
```

urlに関しては変更ないですが、念の為表示しておきます。
accounts/urls.py(一部抜粋)

```
urlpatterns = [
    path('login/', views.CustomLoginView.as_view(), name='login'),
    path('logout/', views.CustomLogoutView.as_view(), name='logout'),

    path('password_change/', views.CustomPasswordChangeView.as_view(), name='password_chan
    path('password_change/done/', views.CustomPasswordChangeDoneView.as_view(), name='pass

    path('password_reset/', views.CustomPasswordResetView.as_view(), name='password_reset'
    path('password_reset/done/', views.CustomPasswordResetDoneView.as_view(), name='passwo
    path('reset/<uidb64>/<token>/', views.CustomPasswordResetConfirmView.as_view(), name='
    path('reset/done/', views.CustomPasswordResetCompleteView.as_view(), name='password_re

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

動作確認

では動作確認をしていきましょう。これまでユーザー名でログインする画面でしたがメールアドレスでログイン出来るようになればOKです。

最後に

Djangoにおける認証の仕組みについて理解が深まれば幸いです。次回はログインユーザーのみが投稿出来るように修正していきます。

3-10. 独自カスタマイズのユーザーを使用する

2019年5月2日　コメントをする

今回のテーマは「独自カスタマイズのユーザーを使用する」です。ここまではDjango標準のUserを使用してきました。しかし標準のUserを使用すると不都合が多い場合も多いです。ユーザーのカスタマイズ方法を見ていきましょう。

※本ページは「3-9. LogoutViewで作成したログアウト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。

usernameの説明文とバリデーションの齟齬の問題

既存ユーザーのusernameのバリデーションはUnicodeUserValidatorが使われいてます。usernameの説明文には「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。」となっています。しかし、実際には漢字やひらがな、カタカナ等の全角文字でもユーザー登録出来てしまいます。これはDjango1.10からpython3のみでusernameでunicodeが許容される仕様に変更されたことによるものです。(参考：リリース1.10)よって既存のユーザーモデルのusernameの説明文は「全角、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。」が正しいということになります。この問題を解決する方法は二つです。１つは説明文やエラーメッセージを変更する方針。もう一つは説明文に合せて半角英数と@/./+/-/_ のみを許容するバリデーションを使用するバリデータに切り替える方針です。今回はバリデータをASCIIUsernameValidatorに変更します。

ユーザーカスタマイズの２つの方法

ユーザーのカスタマイズには大きく分けて二つの方法があります。１つは既存のユーザーモデルの拡張です。例えばプロフィールモデルなどをユーザーのIDとリレーションさせた１対１関係のモデルを用意することで既存ユーザーにない情報を付与することができます。もう一方は独自のユーザーモデルを構築する方法です。この方法の中でもAbstructUserを継承したユーザーモデルを使用する方法と、AbstructBaseUserを継承したユーザーモデルを構築する方法があります。AbstructBaseUserを継承したモデルのほうがカスタマイズ性が高く融通がききます。今回はAbstructBaseUserを継承した独自ユーザーを作成していきます。

データベースの初期化

独自ユーザーモデルを使用する場合は最初のデータベースマイグレーションの前に行う必要があります。そうしないとユーザーに関連する多くのモデルがDjangoの既存ユーザーモデルとリレーションした状態でデータベースが構築されてしまうからです。ここまで、学習の都合で既存Userモデルを使用してきたので、

一度データベースを破棄して新規に作成しましょう。

```
$ mysql -u root mysql
[mysql] drop database forum_data;
[mysql] create database forum_data;
[mysql] grant all on forum_data.* to 'forum_user'@'localhost';
```

独自カスタマイズユーザーの実装

今回は独自ユーザーモデルはaccountsアプリケーション内に作成していきます。
accounts/models.py

```python
from django.db import models
from django.contrib.auth.models import PermissionsMixin
from django.contrib.auth.base_user import AbstractBaseUser, BaseUserManager
from django.contrib.auth.validators import UnicodeUsernameValidator, ASCIIUsernameValidato
from django.core.mail import send_mail
from django.utils.translation import gettext_lazy as _
from django.utils import timezone

# Create your models here.

class UserManager(BaseUserManager):
    """
    Create and save user with email
    """
    use_in_migrations = True

    def _create_user(self, username, email, password, **extra_fields):
        """
        Create and save a user with the given username, email, and password.
        """
        if not username:
            raise ValueError('The given username must be set')

        if not email:
            raise ValueError('The given email must be set')

        email = self.normalize_email(email)
        username = self.model.normalize_username(username)
        user = self.model(username=username, email=email, **extra_fields)
        user.set_password(password)
        user.save(using=self._db)
        return user

    def create_user(self, username, email=None, password=None, **extra_fields):
        extra_fields.setdefault('is_staff', False)
        extra_fields.setdefault('is_superuser', False)
        return self._create_user(username, email, password, **extra_fields)

    def create_superuser(self, username, email, password, **extra_fields):
        extra_fields.setdefault('is_staff', True)
        extra_fields.setdefault('is_superuser', True)

        if extra_fields.get('is_staff') is not True:
            raise ValueError('Superuser must have is_staff=True.')
        if extra_fields.get('is_superuser') is not True:
            raise ValueError('Superuser must have is_superuser=True.')

        return self._create_user(username, email, password, **extra_fields)
```

```
class User(AbstractBaseUser, PermissionsMixin):
    """
    Django標準のUserをベースにカスタマイズしたUserクラス
    """
    username_validator = UnicodeUsernameValidator()
    # python3で半角英数のみ許容する場合はASCIIUsernameValidatorを用いる
    # username_validator = ASCIIUsernameValidator()

    username = models.CharField(
        _('username'),
        max_length=50,
        unique=True,
        # help_text=_('Required. 150 characters or fewer. Letters, digits and @/./+/-/_ on
        help_text='この項目は必須です。全角文字、半角英数字、@/./+/-/_ で50文字以下にしてく
        validators=[username_validator],
        error_messages={
            'unique': _("A user with that username already exists."),
        },
    )
    # first_name = models.CharField(_('first name'), max_length=30, blank=True)
    # last_name = models.CharField(_('last name'), max_length=150, blank=True)
    email = models.EmailField(
        _('email address'),
        help_text='この項目は必須です。メールアドレスは公開されません。',
        blank=False
    )
    is_staff = models.BooleanField(
        _('staff status'),
        default=False,
        help_text=_('Designates whether the user can log into this admin site.'),
    )
    is_active = models.BooleanField(
        _('active'),
        default=True,
        help_text=_(
            'Designates whether this user should be treated as active. '
            'Unselect this instead of deleting accounts.'
        ),
    )
    date_joined = models.DateTimeField(_('date joined'), default=timezone.now)

    objects = UserManager()

    EMAIL_FIELD = 'email'
    USERNAME_FIELD = 'username'
    REQUIRED_FIELDS = ['email']

    class Meta:
        verbose_name = _('user')
        verbose_name_plural = _('users')
```

```
        # abstract = True
        abstract = False

    def clean(self):
        super().clean()
        self.email = self.__class__.objects.normalize_email(self.email)

    # first_nameとlast_nameに関する部分はコメントアウト
    # def get_full_name(self):
    #     """
    #     Return the first_name plus the last_name, with a space in between.
    #     """
    #     full_name = '%s %s' % (self.first_name, self.last_name)
    #     return full_name.strip()

    # def get_short_name(self):
    #     """Return the short name for the user."""
    #     return self.first_name

    def email_user(self, subject, message, from_email=None, **kwargs):
        """Send an email to this user."""
        send_mail(subject, message, from_email, [self.email], **kwargs)
```

基本的にはdjango.contrib.auth.models.UserManagerとdjango.contrib.auth.models.AbstructUserのコピーです。ただし今回はfirst_nameとlast_nameフィールドを削ったので、その部分に関しては変更を加えています。また、バリデータに関してはデフォルトのままUnicodeUserValidatorを使用していますが、もしpython3環境で半角英数を使用したい場合はASCIIUsernameValidatorに変更をしましょう。ただヘルプメッセージは実態に即したメッセージに変更しないと整合性が失われるので変更しています。

独自ユーザー用のフォームの作成

独自のユーザーモデルを作成したので各フォームもそれに従って変更します。多くのものはimportを変更するだけで問題ないですが、ユーザー作成、ユーザー情報変更用のフォームは追加が必要になります。

accounts/forms.py

```python
from django.forms import ModelForm
# from django.contrib.auth.models import User
# from django.contrib.auth import get_user_model
from django.contrib.auth.forms import (
    AuthenticationForm, PasswordChangeForm,
    PasswordResetForm, SetPasswordForm,
    UserChangeForm, UserCreationForm
)

from .models import User

# UserModel = get_user_model()

class UserInfoChangeForm(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()

class EmailChangeForm(ModelForm):
    class Meta:
        model = User
        fields = ['email']

    def __init__(self, email=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

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


class CustomAuthenticationForm(AuthenticationForm):
    def __init__(self, *args, **kwargs):
        kwargs.setdefault('label_suffix', '')
        super().__init__(*args, **kwargs)

class CustomPasswordChangeForm(PasswordChangeForm):
    def __init__(self, *args, **kwargs):
        kwargs.setdefault('label_suffix', '')
        super().__init__(*args, **kwargs)

class CustomPasswordResetForm(PasswordResetForm):
    def __init__(self, *args, **kwargs):
        kwargs.setdefault('label_suffix', '')
        super().__init__(*args, **kwargs)

class CustomSetPasswordForm(SetPasswordForm):
    def __init__(self, *args, **kwargs):
        kwargs.setdefault('label_suffix', '')
        super().__init__(*args, **kwargs)

class AdminUserCreationForm(UserCreationForm):
    class Meta:
        model = User
        fields = ('username', 'email')

    # def save(self, commit=True):
    #     user = User.objects.create_user(
    #         self.cleaned_data["name"],
    #         self.cleaned_data["email"],
    #         self.cleaned_data["password1"],
    #     )
    #     return user

class CustomUserChangeForm(UserChangeForm):
    class Meta:
        model = User
        fields = '__all__'

class CustomUserCreationForm(UserCreationForm):
    class Meta:
        model = User
        fields = ('username', 'email')

    def __init__(self, *args, **kwargs):
```

```
        kwargs.setdefault('label_suffix','')
        super().__init__(*args, **kwargs)
```

少々複雑なので、不要な部分はコメントアウトしています。import部分で先程作成したUserモデルをimportしています。EmailChangeFormはこれまでのUserInfoChangeFormの代わりに使います。ここは大きな問題はないと思います。

ビューの変更

accounts/views.py(一部抜粋)

```python
from django.shortcuts import render, redirect
from django.views.generic import TemplateView, FormView, UpdateView
# from django.contrib.auth.forms import UserCreationForm
# from django.contrib.auth.models import User
from django.contrib.auth.mixins import LoginRequiredMixin
from django.contrib.auth.views import (
    LoginView, LogoutView, PasswordChangeView, PasswordChangeDoneView,
    PasswordResetView, PasswordResetDoneView, PasswordResetConfirmView, PasswordResetCompl
)
from django.urls import reverse_lazy
from django.contrib.auth import login, authenticate

from .models import User
from .forms import (
    UserInfoChangeForm,
    CustomAuthenticationForm, CustomPasswordChangeForm,
    CustomPasswordResetForm, CustomSetPasswordForm,
    CustomUserChangeForm, CustomUserCreationForm, EmailChangeForm
)

# Create your views here.

class UserCreateView(FormView):
    # form_class = UserCreationForm
    form_class = CustomUserCreationForm
    template_name = 'registration/create.html'
    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': 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'))

class UserProfileView(LoginRequiredMixin, TemplateView):
    template_name = 'registration/profile.html'
```

```
    def get_queryset(self):
        return User.objects.get(id=self.request.user.id)

class EmailChangeView(LoginRequiredMixin, FormView):
    template_name = 'registration/change.html'
    form_class = EmailChangeForm
    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,
        })
        return kwargs

class UserChangeView(LoginRequiredMixin, FormView):
    '''
    Django組込みのUserを利用する場合のユーザー情報変更ビュー
    カスタムユーザーでは使用しない
    '''
    template_name = 'registration/change.html'
    form_class = UserInfoChangeForm
    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

class CustomLoginView(LoginView):
    form_class = CustomAuthenticationForm

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

```
class CustomPasswordChangeView(PasswordChangeView):
    form_class = CustomPasswordChangeForm
    template_name = 'registration/password_change_form.html'
    success_url = reverse_lazy('accounts:password_change_done')

class CustomPasswordChangeDoneView(PasswordChangeDoneView):
    template_name = 'registration/password_change_done.html'

class CustomPasswordResetView(PasswordResetView):
    email_template_name = 'registration/password_reset_email.html'
    form_class = CustomPasswordResetForm
    from_email = 'info@example.com'
    subject_template_name = 'registration/password_reset_subject.txt'
    success_url = reverse_lazy('accounts:password_reset_done')
    template_name = 'registration/password_reset_form.html'

class CustomPasswordResetDoneView(PasswordResetDoneView):
    template_name = 'registration/password_reset_done.html'

class CustomPasswordResetConfirmView(PasswordResetConfirmView):
    form_class = CustomSetPasswordForm
    post_reset_login = False
    post_reset_login_backend = None
    success_url = reverse_lazy('accounts:password_reset_complete')
    template_name = 'registration/password_reset_confirm.html'

class CustomPasswordResetCompleteView(PasswordResetCompleteView):
    template_name = 'registration/password_reset_complete.html'
```

ビューに関しては基本的にはUserモデルのインポートとフォームを変更するだけで問題ありません。ただしEmailChangeViewは新設しています。

テンプレートの変更

ユーザーからfirst_nameとlast_nameを削除したため、テンプレートにも多少修正が必要です。以下変更したテンプレートのみ表示します。

templates/accounts/profile.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 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}}
                    <div class="item">名前：{% if user.first_name %}{{user.first_name}}{%
                </div>
            </div>
        </div>
        <a class="ui button" href="{% url 'accounts:change' %}">登録情報変更</a>
        <a class="ui button" href="{% url 'accounts:password_change' %}">パスワード変更</a
    </div>
    {% include 'base/sidebar.html' %}
</div>
{% endblock %}
```

templates/accounts/create_confirm.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 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>メールアドレス</td><td>{{form.cleaned_data.email}}</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 %}
```

独自ユーザーを管理画面で表示する設定

独自に作成したユーザーは登録しないと管理画面で表示・修正することは出来ません。
accounts/admin.pyに追記します。

accounts/admin.py

```
from django.contrib import admin
from django.contrib.auth.admin import UserAdmin
from django.utils.translation import gettext_lazy as _

from .models import User
from .forms import AdminUserCreationForm, CustomUserChangeForm

class CustomUserAdmin(UserAdmin):
    fieldsets = (
        (None, {'fields': ('username', 'password')}),
        (_('Personal info'), {'fields': ('email',)}),
        (_('Permissions'), {'fields': ('is_active', 'is_staff', 'is_superuser', 'groups',
        (_('Important dates'), {'fields': ('last_login', 'date_joined')}),
                                                )
    add_fieldsets = ((None, {'classes': ('wide',), 'fields': ('username', 'email', 'passwo
    form = CustomUserChangeForm
    add_form = AdminUserCreationForm
    list_display = ('username', 'email','is_staff')
    search_fields = ('username', 'email')
    ordering = ('username',)

admin.site.register(User, CustomUserAdmin)
```

ここも基本的にはdjango.contrib.auth.admin.UserAdminを継承したクラスを作成してパラメータを上書きしただけです。特にformとadd_formに作成したフォームを指定することを忘れないで下さい。
add_formはUserモデルを作成するためのフォームです。間違えないようにしましょう。

URL設定

ビューに変更を加えているのでURL設定も変更しましょう。

accounts/urls.py

```
from django.urls import path, include
from django.contrib.auth import views as av
from . import views
from .forms import (
    CustomAuthenticationForm, CustomPasswordChangeForm
)

app_name = 'accounts'

urlpatterns = [
    # path('', include('django.contrib.auth.urls')),
    # copy from django.contrib.auth.urls.py
    # path('login/', av.LoginView.as_view(form_class=CustomAuthenticationForm
    #                                     ), name='login'),
    path('login/', views.CustomLoginView.as_view(), name='login'),
    path('logout/', views.CustomLogoutView.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(
    #     form_class=CustomPasswordChangeForm,
    #     template_name='registration/password_change_form.html',
    #     success_url='accounts/password_change/done/'
    # ), name='password_change'),
    # path('password_change/done/', av.PasswordChangeDoneView.as_view(
    #     template_name = 'registration/password_change_done.html',
    # ), name='password_change_done'),

    path('password_change/', views.CustomPasswordChangeView.as_view(), name='password_chan
    path('password_change/done/', views.CustomPasswordChangeDoneView.as_view(), name='pass


    # 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('password_reset/', views.CustomPasswordResetView.as_view(), name='password_reset'
    path('password_reset/done/', views.CustomPasswordResetDoneView.as_view(), name='passwo
    path('reset///', views.CustomPasswordResetConfirmView.as_view(), name='password_reset_
    path('reset/done/', views.CustomPasswordResetCompleteView.as_view(), name='password_re


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

基本的にはEmailChangeViewを設定するだけです。

独自ユーザーを使用する設定

独自ユーザーを使用するためにはAUTH_USER_MODELを指定する必要があります。この値はデフォルトでは'auth.User'となっているのですが、この値を{アプリケーション名}.{ユーザーモデル名}に変更します。
mysite/settings.py(一部抜粋)

```
+ AUTH_USER_MODEL = 'accounts.User'
```

カスタムユーザーモデルをThreadアプリのモデルに適用する

ここまで学習のためThreadアプリのモデルにはDjangoのユーザーを適用させてきました。カスタムユーザーを作成したのでThreadモデルにもカスタムユーザーを紐付ける修正をしましょう。
thread/models.py(一部抜粋)

```
from accounts.models import User
・・・省略・・・

class Topic(models.Model):
    ・・・省略・・・
    user = models.ForeignKey(
        User,
        on_delete=models.SET_NULL,
        null=True,
        blank=True,
    )
    email = models.EmailField(
        verbose_name='メールアドレス',
        null=True,
        blank=True,
    )
    ・・・省略・・・

class Comment(models.Model):
    ・・・省略・・・
    user = models.ForeignKey(
        User,
        on_delete=models.SET_NULL,
        null=True,
        blank=True,
    )
    email = models.EmailField(
        verbose_name='メールアドレス',
        null=True,
        blank=True,
    )
    ・・・省略・・・
```

データベースマイグレーション

では、新規作成したデータベースにマイグレーションを行いましょう。以下コマンドを入力してマイグレーションを行います。

```
$ ./manage.py makemigrations
$ ./manage.py migrate
```

動作確認

では動作確認をしていきましょう。

ユーザー登録画面

ユーザープロフィール画面

ユーザー情報変更画面

最後に

今回はほとんど、Django既存ユーザーのコピーを追加した感じでしたが、このユーザーをベースにより自分好みのモデルにカスタマイズしていくことが出来ます。次回はログイン方式を変更していきましょう。