Django基本講座４
(viewの応用)

Viewの処理応用（リダイレクト、エラーハンドリング）

 # リダイレクト(redirect(from django.shortcuts import redirect))
リダイレクトをすると、特定のURLに処理を移すことができる
return redirect('https://www.google.com') **# googleに画面遷移する**
return redirect('app:sample') **# app_nameがappでnameがsampleの関数に遷移する**
return redirect('app:sample', id=1) **# app_nameがappでnameがsampleの関数にid=1を引数として遷移する**

Viewの処理応用（リダイレクト、エラーハンドリング）

 # エラーハンドラー
404や500などのエラーが発生した場合に特定のViewに処理を渡したい場合に用いられます。
このようにすることで、エラーが発生した場合にエラー画面をユーザーに表示しないようにすることができます。

400エラー	不正な構文、無効なリクエストメッセージフレーミング、または不正なリクエストルーティングのために、サーバーがクライアントによって送信されたリクエストを処理できなかったことを示す
403エラー	ユーザーにアクセス権がなく閲覧禁止となっていることを示す
404エラー	URLに対応するページが存在しないことを示す
500エラー	インターナルサーバエラー。サーバ側の処理に問題があることを示す

Viewの処理応用（リダイレクト、エラーハンドリング）

 # エラーハンドリングの方法
settings.py
デバッグモードをFalseに設定する(DEBUGがTrueの場合エラーハンドリングはできずにそのままエラー画面が表示される)
DEBUG = True → DEBUG = False

ALLOWED_HOSTSに自身のホストを追加する(DEBUGモードでない場合はホストの追加が必要)
ALLOWED_HOSTS = ['127.0.0.1']

プロジェクトのurls.pyに設定したいステータスコードのハンドラーを追加する

handler404 = views.関数名(404エラーの場合に実行したい関数を指定(関数には引数を2つとる))
handler500 = views.関数名(500エラーの場合に実行したい関数を指定(関数には引数を1つとる))

 # 意図的に404エラーを発生させる方法
from django.http import Http404
raise Http404 # 404エラーを発生させたい場所で実行する
参考: https://docs.djangoproject.com/ja/3.1/ref/views/#error-views

Viewの処理応用（リダイレクト、エラーハンドリング）

 # get_object_or_404(django.shortcuts.get_object_or_404)
指定したモデルを呼び出し、getを行う。値を取得できなかった場合、raise Http404を送出する
例)
get_object_or_404(Book, title__startswith='M', pk=1)

 # get_list_or_404(django.shortcuts.get_list_or_404)
指定したモデルを呼び出し、filterを行う。値を取得できなかった場合、raise Http404を送出する
例)
my_objects = get_list_or_404(MyModel, published=True)

Viewの処理応用（ログイン機能の実装）

```
INSTALLED_APPS = [
    :
   ‘django.contrib.auth‘, # ユーザ情報の認証に用いられる
    :
]
```

ユーザ情報を保存する際に、パスワードは、必ずハッシュ化して保存する。

ハッシュ化する関数を指定する場合には、settings.pyのPASSWORD_HASHERS変数を追加する
https://docs.djangoproject.com/ja/3.1/topics/auth/passwords/#included-hashers

ハッシュアルゴリズムは、PASSWORD_HASHERSの中のリストの上から順に利用できるものが利用される。上位には、ArgonやBcryptなどの強力なアルゴリズムを置くと良い

パスワードのバリデーション
https://docs.djangoproject.com/ja/3.1/topics/auth/passwords/#enabling-password-validation

Viewの処理応用（ログイン機能の実装）

django.contrib.auth.models.User: ログイン用のユーザとして利用する(管理画面のログインにも用いられるが、これを用いてウェブサイトのログイン、ログアウトを行う)
https://docs.djangoproject.com/ja/3.1/ref/contrib/auth/#user-model

デフォルトのフィールドにフィールドを追加する場合、OneToOneFiledで紐づけたモデルを作成すればよい

```
class UserProfile(models.Model):
    user = models.OneToOneField(User, on_delete=models.CASCADE)
   :
```

user.set_password(パスワード): ユーザのレコードにパスワードを設定できる

Viewの処理応用（ログイン機能の実装）

LOGIN_URL: ログインに利用するURLを指定する(settings.pyに記述)

ビューで利用する処理
django.contrib.auth.authenticate(username, password): 引数に名前とパスワードを取り、ユーザが存在して、パスワードが正しいかチェックする
django.contrib.auth.login(request, user): ログインを行う
django.contrib.auth.logout: ログアウトを行う
django.contrib.auth.decorators.login_required: ログインが必要な関数にデコレータとして付与すると、ログインしていない場合にはエラーにすることができる。

テンプレートで利用する処理
{% if user.is_authenticated %}: ログインしている場合だけ、実行される。

ユーザ作成とログイン・ログアウトに関して、もう少しカスタマイズの方法もありますが、それは次にご説明いたします。

ログイン機能の実装（ユーザとパスワードのバリデーション）

パスワードが妥当なものかを判断するには
django.contrib.auth.password_validation.validate_password,
django.contrib.auth.password_validation.password_changedを用いる。
(https://docs.djangoproject.com/ja/3.1/topics/auth/passwords/#integrating-validation)

validate_password(password, user): ユーザのパスワードが適切か（短すぎなくないか、ありきたり過ぎないか、ユーザ名から類推が容易でないか等）チェックをする

パスワードが適切でない場合には、バリデーションエラーが発生する

```
try:
    validate_password(user_form.cleaned_data.get('password', user))
except ValidationError as e:
    return render(request, ‘template.html’)
```

ログイン機能の実装（バリデーターの追加）

パスワードのバリデーターは自分で作成して、追加することもできる。
(https://docs.djangoproject.com/ja/3.1/topics/auth/passwords/#writing-your-own-validator)

クラスを作成して、中に__init__, validate, get_help_textを定義する。

```
class MinimumLengthValidator:
    def __init__(self, min_length=8):
        self.min_length = min_length

    def validate(self, password, user=None):
        pass

    def get_help_text(self):
        pass
```

次に、settings.pyのAUTH_PASSWORD_VALIDATORSで、作成したバリデーターを指定する。

今回は、正規化ライブラリreを用います。
https://docs.python.org/ja/3/library/re.html

ログイン機能の実装（ユーザークラスのカスタマイズ）

ユーザーのクラスを利用する場合には、元のユーザーをカスタマイズをして、用いることが多い
この際に、AbstractUserかAbstractBaseUserを用いる。
AbstractUser: すでに存在するフィールドをそのまま流用して、usernameフィールドを削除したい場合用いるとよい
AbstractBaseUser: 初めからUserを作り変えたい（今回はこちらを利用する）

以下の手順で作成する。
1. カスタムマネージャーとカスタムユーザーのクラスを作成する
2. settings.pyを修正して、ユーザーはカスタムのクラスを指すようにする
3. マイグレーションを行う
4. フォームやAdminを作成する

https://docs.djangoproject.com/ja/3.1/topics/auth/customizing

一般的なユーザーは
https://docs.djangoproject.com/ja/3.1/ref/contrib/auth/#user-model

また、カスタムのユーザーにsuperuserなどの一般的な、Djangoのパーミッションを取り入れるには、PermissionsMixin(https://docs.djangoproject.com/ja/3.1/topics/auth/customizing/#custom-users-and-permissions)を用いると良い

ログイン機能の実装（ユーザークラスのカスタマイズ）

実装するには、以下の実装例がわかりやすい。
https://docs.djangoproject.com/ja/3.1/topics/auth/customizing/#a-full-example

1.　カスタムマネージャーとカスタムユーザーを作成する
カスタムマネージャーは、django.contrib.auth.models.BaseUserManagerを継承して作成し、create_userとcreate_superuserメソッドを追加して、ユーザ作成時、スーパーユーザー作成時の処理を追加する。

カスタムユーザーは、 django.contrib.auth.models.AbstractBaseUserを継承して作成し、必要なフィールド情報を記入する。

2. settings.pyを修正して、ユーザーはカスタムのクラスを指すようにする
AUTH_USER_MODEL = 'users.CustomUser’

3. マイグレーションを行う(makemigrations → migrate)
4. フォームやAdminを作成する
管理画面からユーザーを登録変更するために、Adminの中身を変更する

Viewの処理応用（管理画面のカスタマイズ）

classの中に__str__(self)を定義することでそのクラスのレコードの表示を変えることができる

```
class Model(models.Model):
    :
    def __str__(self):
        ….
```

各モデルのページをカスタマイズするには、admin.ModelAdminを継承したクラスを作成して、中に内容を定義し、registerの際に第２引数に取る

```
class MyAdmin(admin.ModelAdmin):
    :

admin.site.register(Model, MyAdmin)
```

または、

```
@admin.register
class MyAdmin(admin.ModelAdmin)
```

Viewの処理応用（管理画面のカスタマイズ）

adminの追加する要素

fields: 編集画面で表示するフィールドの順番を変更する。
search_fields: 検索に利用したいフィールドを記述する。
list_filter: 特定のフィールドでフィルターをできるようにする。
list_display: 一覧画面で表示するフィールドを指定する。
list_display_links: 編集画面への遷移をするリンクに指定するフィールドを変更する。
list_editable: 一覧画面で編集できるようにするフィールドを指定する。

modelsに追加する要素
class Metaに以下のことを追加することで、表示内容を変えることができる
ordering: 画面の並びを変える
verbose_name_plural: 管理画面の表示を変える

テンプレートを利用して、管理画面を上書き修正する。
https://github.com/django/django/tree/master/django/contrib/admin/templates
admin: 管理画面ページ一般
registration: ログアウト、パスワード変更等