Aplikacje WWW. Wykład #3

Praca z modelami we frameworku Django 5.2

1. Definiowanie modeli

> Dokumentacja modeli: https://docs.djangoproject.com/pl/5.2/topics/db/models/ Dokumentacja dostępnych typów pól: https://docs.djangoproject.com/pl/5.2/ref/models/fields/#model-field-types

Poniżej znajduje się lista dostepnych typów oraz przyklady ich definicji.

1. **CharField**

Przechowuje tekst o ograniczonej długości.

- **Przykład**: Imię, nazwisko, tytuł książki.
- **Użycie**:

```
from django.db import models

class Person(models.Model):
    first_name = models.CharField(max_length=50)
    last_name = models.CharField(max_length=50)
```

2. **TextField**

Przechowuje długi tekst, bez limitu długości (przydatne dla treści jak opisy, artykuły).

- **Przykład**: Opis produktu, artykuł na blogu.
- **Użycie**:

```
class Article(models.Model):
    title = models.CharField(max_length=100)
    content = models.TextField()
```

3. **IntegerField**

Przechowuje liczby całkowite.

- **Przykład**: Wiek, liczba przedmiotów.
- **Użycie**:

```
class Product(models.Model):
    name = models.CharField(max_length=100)
    stock_quantity = models.IntegerField()
```

4. **FloatField**

Przechowuje liczby zmiennoprzecinkowe.

- **Przykład**: Cena produktu, współrzędne geograficzne.
- **Użycie**:

```
class Product(models.Model):
    name = models.CharField(max_length=100)
    price = models.FloatField()
```

5. **BooleanField**

Przechowuje wartość logiczną (`True` lub `False`).

- **Przykład**: Czy produkt jest dostępny.
- **Użycie**:

```
class Product(models.Model):
    name = models.CharField(max_length=100)
    is_available = models.BooleanField(default=True)
```

6. **DateField**

Przechowuje daty.

- **Przykład**: Data urodzenia, data publikacji.
- **Użycie**:

```
class Event(models.Model):
    event_name = models.CharField(max_length=100)
    event_date = models.DateField()
```

7. **DateTimeField**

Przechowuje daty i godziny.

- **Przykład**: Data i czas rejestracji użytkownika, czas utworzenia posta.
- **Użycie**:

```
class Post(models.Model):
    title = models.CharField(max_length=100)
    created_at = models.DateTimeField(auto_now_add=True)
```

8. **EmailField**

Specjalizacja `CharField` przechowująca adresy email.

- **Przykład**: Email użytkownika.
- **Użycie**:

```
class User(models.Model):
    name = models.CharField(max_length=100)
    email = models.EmailField(unique=True)
```

9. **URLField**

Przechowuje adresy URL.

- **Przykład**: Strona internetowa firmy.
- **Użycie**:

```
class Company(models.Model):
    name = models.CharField(max_length=100)
    website = models.URLField()
```

10. **ForeignKey**

Tworzy relację jeden-do-wielu (One-to-Many) z inną tabelą.

- **Przykład**: Posty powiązane z użytkownikiem.
- **Użycie**:

```
class User(models.Model):
    name = models.CharField(max_length=100)

class Post(models.Model):
    title = models.CharField(max_length=100)
    content = models.TextField()
    author = models.ForeignKey(User, on_delete=models.CASCADE)
```

11. **ManyToManyField**

Tworzy relację wiele-do-wielu (Many-to-Many) z inną tabelą.

- **Przykład**: Grupy użytkowników w systemie.
- **Użycie**:

```

```

```
class Group(models.Model):
    name = models.CharField(max_length=100)

class User(models.Model):
    name = models.CharField(max_length=100)
    groups = models.ManyToManyField(Group)
```

12. **OneToOneField**

Tworzy relację jeden-do-jednego (One-to-One) z inną tabelą.

- **Przykład**: Profil użytkownika.
- **Użycie**:

```
class User(models.Model):
    name = models.CharField(max_length=100)

class Profile(models.Model):
    user = models.OneToOneField(User, on_delete=models.CASCADE)
    bio = models.TextField()
```

13. **DecimalField**

Przechowuje liczby dziesiętne z określoną precyzją (przydatne dla walut, wartości finansowych).

- **Przykład**: Cena produktu.
- **Użycie**:

```
class Product(models.Model):
    name = models.CharField(max_length=100)
    price = models.DecimalField(max_digits=10, decimal_places=2)
```

14. **SlugField**

Przechowuje krótkie etykiety (slug), często używane w adresach URL.

- **Przykład**: Etykieta dla artykułu.
- **Użycie**:

```
class Article(models.Model):
    title = models.CharField(max_length=100)
    slug = models.SlugField(unique=True)
```

15. **FileField**

Przechowuje ścieżkę do pliku (np. dokumentu lub obrazu).

- **Przykład**: Załączniki do wiadomości, zdjęcia użytkowników.
- **Użycie**:

```
class Document(models.Model):
    title = models.CharField(max_length=100)
    file = models.FileField(upload_to='documents/')
```

16. **ImageField**

Specjalizacja `FileField` dla plików graficznych.

- **Przykład**: Awatar użytkownika.
- **Użycie**:

```
class UserProfile(models.Model):
    user = models.OneToOneField(User, on_delete=models.CASCADE)
    avatar = models.ImageField(upload_to='avatars/')
```

17. **TimeField**

Przechowuje czas (bez daty).

- **Przykład**: Czas rozpoczęcia spotkania.
- **Użycie**:

```
class Meeting(models.Model):
    title = models.CharField(max_length=100)
    start_time = models.TimeField()
```

18. **DurationField**

Przechowuje czas trwania.

- **Przykład**: Czas trwania filmu, sesji.
- **Użycie**:

```
class Movie(models.Model):
    title = models.CharField(max_length=100)
    duration = models.DurationField()
```

Każde pole w modelu jest przetwarzane i walidowane automatycznie przez Django ORM, co znacznie upraszcza pracę z danymi. Można dodatkowo używać atrybutów takich jak `null=True`, `blank=True`,

`default=...`, aby dostosować sposób przechowywania danych w bazie.

2. Atrybuty modeli

> Dokumentacja: https://docs.djangoproject.com/pl/5.2/topics/db/models/#field-options

Atrybuty modeli w Django definiują różne aspekty zachowania pól i modeli w interakcji z bazą danych oraz walidacją. Oto najczęściej używane atrybuty modeli Django, ich przykłady i zastosowanie:

1. `null`

- Określa, czy pole może przechowywać wartość `NULL` w bazie danych.
- **Typ pól**: Wszystkie pola (oprócz `ManyToManyField`, `OneToOneField` i `ForeignKey`).
- **Domyślna wartość**: `False` (pole musi mieć wartość).
- **Przykład**:

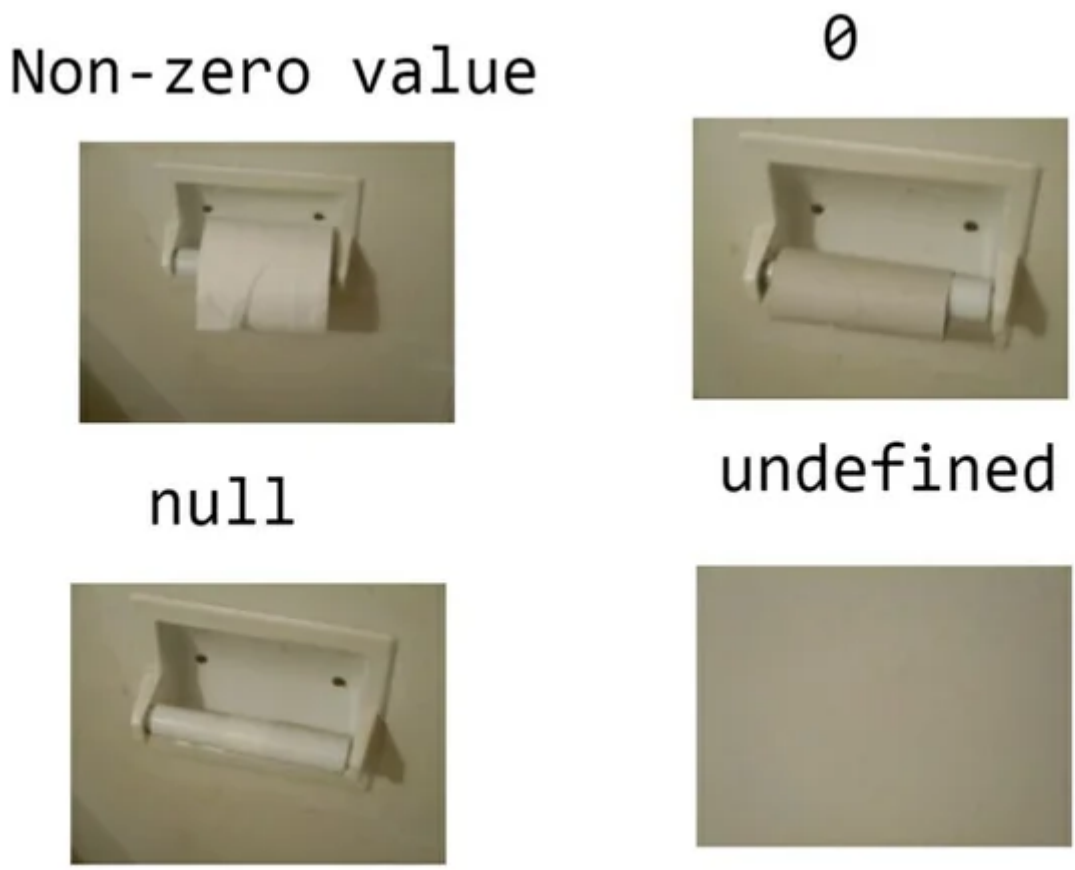
```
class Person(models.Model):
    name = models.CharField(max_length=100, null=True)
```

2. `blank`

- Określa, czy pole może być puste podczas walidacji formularzy (dotyczy walidacji na poziomie formularzy, a nie bazy danych).
- **Typ pól**: Wszystkie pola.
- **Domyślna wartość**: `False` (pole nie może być puste w formularzach).
- **Przykład**:

```
class Person(models.Model):
    email = models.EmailField(blank=True)
```

Jaka jest różnica?



3. `default`

- Ustala domyślną wartość dla pola, jeśli użytkownik jej nie poda.
- **Typ pól**: Wszystkie pola.
- **Przykład**:

```
class Product(models.Model):
    price = models.DecimalField(max_digits=10, decimal_places=2,
default=0.00)
```

4. `unique`

- Sprawia, że wartość w danym polu musi być unikalna w całej tabeli.
- **Typ pól**: Wszystkie pola (oprócz pól relacyjnych jak `ManyToManyField`).
- **Przykład**:

```
class User(models.Model):
    username = models.CharField(max_length=50, unique=True)
```

5. `choices`

- Definiuje listę dostępnych wartości dla pola (jako pary: wyświetlana wartość i wewnętrzna wartość).

- **Typ pól**: Najczęściej `CharField`, `IntegerField`, itp.
- **Przykład**:

```
class Person(models.Model):
    STATUS_CHOICES = [
        ('A', 'Active'),
        ('I', 'Inactive'),
    ]
    status = models.CharField(max_length=1, choices=STATUS_CHOICES,
default='A')
```

6. `primary_key`

- Określa, czy pole jest kluczem głównym (zastępuje automatyczne tworzenie pola `id`).
- **Typ pól**: Zazwyczaj `IntegerField` lub `CharField`.
- **Przykład**:

```
class Person(models.Model):
    ssn = models.CharField(max_length=9, primary_key=True)
```

7. `db_index`

- Tworzy indeks na kolumnie, co przyspiesza operacje wyszukiwania.
- **Typ pól**: Wszystkie pola.
- **Przykład**:

```
class Product(models.Model):
    sku = models.CharField(max_length=30, db_index=True)
```

8. `verbose_name`

- Określa bardziej opisową nazwę pola, wyświetlaną w panelu administracyjnym Django.
- **Typ pól**: Wszystkie pola.
- **Przykład**:

```
class Person(models.Model):
    first_name = models.CharField(max_length=50, verbose_name="First Name")
```

9. `verbose_name_plural`

- Definiuje liczbę mnogą nazwy modelu, wyświetlaną w panelu administracyjnym.
- **Typ modeli**: Wszystkie modele.
- **Przykład**:

```
class Person(models.Model):
    name = models.CharField(max_length=100)

    class Meta:
        verbose_name_plural = "People"
```

10. `help_text`

- Dodaje pomocniczy tekst wyświetlany w formularzach, opisujący pole.
- **Typ pól**: Wszystkie pola.
- **Przykład**:

```
class Product(models.Model):
    price = models.DecimalField(max_digits=10, decimal_places=2,
help_text="Enter price in USD")
```

11. `auto_now`

- Automatycznie ustawia pole na bieżący czas i datę przy każdej aktualizacji obiektu.
- **Typ pól**: `DateField`, `DateTimeField`.
- **Przykład**:

```
class Post(models.Model):
    updated_at = models.DateTimeField(auto_now=True)
```

12. `auto_now_add`

- Automatycznie ustawia pole na bieżący czas i datę przy tworzeniu obiektu.
- **Typ pól**: `DateField`, `DateTimeField`.
- **Przykład**:

```
class Post(models.Model):
    created_at = models.DateTimeField(auto_now_add=True)
```

13. `on_delete`

- Określa zachowanie relacyjnych pól (np. `ForeignKey`) przy usunięciu obiektu, do którego odnosimy się.
- **Typ pól**: `ForeignKey`, `OneToOneField`.
- **Przykład**:

```
```

```
class Post(models.Model):
    author = models.ForeignKey(User, on_delete=models.CASCADE)
```

- Najczęściej używane wartości dla `on_delete`:
  - `CASCADE`: Usuwa powiązane obiekty.
  - `SET_NULL`: Ustawia `NULL` (pole musi mieć `null=True`).
  - `PROTECT`: Blokuje usunięcie powiązanego obiektu.
  - `SET_DEFAULT`: Ustawia wartość domyślną (pole musi mieć `default`).
  - `DO_NOTHING`: Nie wykonuje żadnej akcji.

14. `related_name`

- Określa nazwę odwrotnej relacji dla pól relacyjnych (`ForeignKey`, `ManyToManyField`, `OneToOneField`).
- **Typ pól**: `ForeignKey`, `OneToOneField`, `ManyToManyField`.
- **Przykład**:

```
class Book(models.Model):
    title = models.CharField(max_length=100)

class Author(models.Model):
    name = models.CharField(max_length=100)
    books = models.ManyToManyField(Book, related_name='authors')
```

15. `unique_together`

- Definiuje zestaw pól, które muszą być unikalne razem (np. kombinacja pól).
- **Typ modeli**: Meta klasy modeli.
- **Przykład**:

```
class Order(models.Model):
    customer = models.ForeignKey(User, on_delete=models.CASCADE)
    order_date = models.DateField()

    class Meta:
        unique_together = ('customer', 'order_date')
```

16. `ordering`

- Określa domyślne sortowanie rekordów w zapytaniach do bazy danych.
- **Typ modeli**: Meta klasy modeli.
- **Przykład**:

```
class Product(models.Model):
    name = models.CharField(max_length=100)
```

```
    price = models.DecimalField(max_digits=10, decimal_places=2)

    class Meta:
        # rosnąco
        ordering = ['price']

        # malejąco
        # ordering = ['-price']
```

17. `editable`

- Określa, czy pole może być edytowane w formularzach Django (np. panelu administracyjnym).
- **Typ pól**: Wszystkie pola.
- **Przykład**:

```
class Product(models.Model):
    sku = models.CharField(max_length=30, editable=False)
```

18. `limit_choices_to`

- Ogranicza wybory dla pól relacyjnych, np. `ForeignKey` lub `ManyToManyField`, na podstawie filtrów.
- **Typ pól**: `ForeignKey`, `ManyToManyField`.
- **Przykład**:

```
class ActiveManager(models.Manager):
    def get_queryset(self):
        return super().get_queryset().filter(is_active=True)

class Product(models.Model):
    name = models.CharField(max_length=100)
    manager = models.ForeignKey('self', limit_choices_to={'is_active':
True}, on_delete=models.CASCADE)
```

3. Klasa `Meta`

Wszystkie możliwości klasy `Meta` są opisane w dokumentacji pod adresem:
https://docs.djangoproject.com/pl/5.2/ref/models/options/#

Najważniejsze opcje (część z nich pojawila się w przykładach powyżej):

- `db_table` – nazwa tabeli w bazie danych (domyślnie generowana automatycznie).
- `ordering` – domyślne sortowanie rekordów, np. `ordering = ['name']`.
- `verbose_name` – czytelna nazwa modelu (liczba pojedyncza).
- `verbose_name_plural` – czytelna nazwa modelu (liczba mnoga).
- `unique_together` – zestaw pól, które muszą być unikalne razem (jako krotka lub lista krotek).
- `index_together` – zestaw pól, dla których tworzony jest wspólny indeks.

- `constraints` – lista dodatkowych ograniczeń (np. `CheckConstraint`, `UniqueConstraint`).
- `permissions` – lista niestandardowych uprawnień.
- `default_related_name` – domyślna nazwa relacji odwrotnej.
- `get_latest_by` – pole używane przez `Model.objects.latest()`.
- `managed` – czy Django zarządza tabelą (domyślnie `True`).
- `app_label` – ręczne przypisanie modelu do aplikacji (gdy model jest poza folderem aplikacji).
- `default_permissions` – domyślne uprawnienia generowane przez Django (np. ('add', 'change', 'delete', 'view')).

4. Przykłady

1. Relacje między modelami

Przykład relacji wiele do wielu.

> Dokumentacja: https://docs.djangoproject.com/pl/5.2/topics/db/models/#many-to-many-relationships

Listing 1

```
# plik models.py
from django.db import models


class Category(models.Model):
    name = models.CharField(max_length=30)
    description = models.TextField(null=True, blank=True)

    class Meta:
        verbose_name_plural = 'categories'


class Product(models.Model):
    name = models.CharField(max_length=200)
    description = models.TextField(null=True, blank=True)
    price = models.DecimalField(max_digits=6, decimal_places=2)
    categories = models.ManyToManyField(Category)
```

W bazie danych zostanie stworzona dodatkowa tabela realizująca złączanie wiele do wielu. Pole definiujemy tylko w jednej klasie modelu, nie ma znaczenia w której.

Możliwe jest również zdefiniowanie relacji wiele do wielu dla "samego siebie". Moglibyśmy to zadeklarować dla modelu `Category`, aby umożliwić przypisywanie wielu kategorii dla dowolnej innej kategorii (pomijając aktualnie sens takiego podejścia). Nazywane jest to tutaj relacją rekurencyjną (https://docs.djangoproject.com/pl/5.2/ref/models/fields/#recursive-relationships). Parametr `symmetrical` służy do określania czy ta relacja ma być automatycznie symetryczna. Jeżeli `symmetrical=True` oznacza to dodanie dwóch wierszy do tabeli. Tu rezygnujemy z takiej funkcjonalności.

Listing 2

```
# plik models.py
from django.db import models


class Category(models.Model):
    name = models.CharField(max_length=30)
    description = models.TextField(null=True, blank=True)
    subcategory = models.ManyToManyField("self", symmetrical=False)

    class Meta:
        verbose_name_plural = 'categories'


class Product(models.Model):
    name = models.CharField(max_length=200)
    description = models.TextField(null=True, blank=True)
    price = models.DecimalField(max_digits=6, decimal_places=2)
```

Relacja rekurencyjna do samego siebie typu jeden do jeden

Bardziej sensowne wydaje się wykorzystanie relacji jeden do jeden dla modelu `Category`, aby ustawić kategorię nadrzędną.

Listing 3

```
# plik models.py
from django.db import models


class Category(models.Model):
    name = models.CharField(max_length=30)
    description = models.TextField(null=True, blank=True)
    parent_category = models.ForeignKey("self", null=True,
on_delete=models.RESTRICT)

    class Meta:
        verbose_name_plural = 'categories'


class Product(models.Model):
    name = models.CharField(max_length=200)
    description = models.TextField(null=True, blank=True)
    price = models.DecimalField(max_digits=6, decimal_places=2)
```

Więcej przykładów znajdziesz w oficjalnej dokumentacji:

- https://docs.djangoproject.com/pl/5.2/topics/db/examples/many_to_one/

- https://docs.djangoproject.com/pl/5.2/topics/db/examples/many_to_many/
- https://docs.djangoproject.com/pl/5.2/topics/db/examples/one_to_one/