Formularze HTML

Formularz : struktura formularza

```
<form>
 |-- <???>
      |-- <fieldset>                  zbiór pół
      |    |-- <legend>                opis zbioru pół
      |    |-- <input>                 pole danych
      |-- <fieldset>                  zbiór pół
           |-- <legend>                opis zbioru pół
           |-- <input>                 pole danych
           |-- <input type="submit"> przycisk
```

Wymagany jest jeden element blokowy (np. *div*, *table*, *fieldset*) obejmujący całą zawartość formularza. Aby formularz można było wysłać do serwera, niezbędny jest element typu *submit*. Formularz powinien też zawierać pola danych, np. typu *input*.

```
<form>
 |-- <fieldset>                       zbiór pół
      |-- <legend>                     opis zbioru pół
      |-- <input type="text">          pole edycyjne
      |-- <input type="submit">        przycisk
```

W najprostszym przypadku wystarczy jeden element *fieldset*, opcjonalnie uzupełniony przez *legend*, pola danych (np. *input*) oraz przycisk wysyłania formularza *submit*.

Formularz: form

- Wstawia do dokumentu formularz

```
<form method="POST" action="/adres/url/">
  ...
</form>
```

- Znacznik zamykający – obowiązkowy

Atrybuty

- action – formularze typowo są obsługiwane po stronie serwera, dlatego w atrybucie *action* trzeba podać URL skryptu (lub innego typu dynamicznej strony WWW), do którego zostanie wysłany formularz;
- method – sposób wysłania formularza do serwera; formularz może być wysłany przez HTTP przy pomocy żądania POST albo GET – różnica polega na sposobie przekazywania parametrów:
  - POST – w nagłówkach HTTP
  - GET – w adresie WWW, zatem są widoczne dla użytkownika, a także zapisywane w historii przeglądarki; Wybór metody zależy od sytuacji: obie mają wady i zalety.
- enctype – rodzaj zawartości formularza
  - jeżeli formularz nie zawiera plików: `application/x-www-form-urlencoded` (jest to wartość domyślna – można pominąć)
  - jeżeli formularz zawiera dołączone pliki: `multipart/form-data`

- accept – dotyczy plików dołączanych do formularza; jest to lista typów zawartości, jakie są akceptowane przez docelowy skrypt (wskazany atrybutem *action*) – przeglądarka może wykorzystać ten atrybut podczas wyświetlania okna wyboru pliku

Zbiór pól: fieldset i legend

- fieldset – reprezentuje zbiór powiązanych ze sobą elementów formularza; wyświetlany jako owalna ramka, z opcjonalnym podpisem w postaci elementu legend
- legend – wyświetla opis dla elementu fieldset

```
<form>
  <fieldset>
    <legend>Tytuł</legend>
    <input …>
  </fieldset>
</form>
```

- Znacznik zamykający – obowiązkowy

Pola danych formularzy HTML:

- Pola tekstowe, jedno- i wielolinijkowe
- Przyciski
  - przycisk *submit* – wysyła zawartość formularza do serwera,
  - przycisk *reset* – przywraca wartości początkowe danych,
  - inne przyciski – działanie zależne od skojarzonego skryptu, wykonywanego po stronie przeglądarki
- Pola wyboru
- Przyciski radiowe
- Listy rozwijane
- Pola wyboru pliku
- Pola ukryte
- Obiekty osadzone

Pole danych: input

- Reprezentuje pole danych:

```
<input type="…" name="…">
```

- Znacznik zamykający – zabroniony
- Atrybuty:
  id, class, style, title,
  type – typ pola (np. text, checkbox, radio, submit, …)
  name – nazwa, pod jaką zawartość będzie wysłana do serwera
  value – wartość początkowa (znaczenie zależnie od typu)
  checked – pole zaznaczone (dla typu checkbox i radio)
  disabled – niedostępne dla użytkownika
  readolny – pole tylko do odczytu
  required – pole wymagane
  size – rozmiar pola (znaczenie zależnie od typu)
  maxlength – maksymalna długość tekstu (dla text i password)
  src – URL pliku z obrazkiem (dla typu image)

tabindex – kolejność aktywacji klawiszem tabulatora
accesskey – klawisz szybkiego dostępu (skrót klawiaturowy – niestety sposób aktywacji zależy od systemu operacyjnego)
onfocus – skrypt wykonywany po uzyskaniu "fokusu"
onblur – skrypt wykonywany po utracie "fokusu"
onselect – skrypt wykonywany zmianie zaznaczenia tekstu
onchange – skrypt wykonywany modyfikacji tekstu

Pole danych: atrybuty type

- Określa typ kontrolki, jaka ma zostać utworzona:
  - text – pole edycyjne, jednolinijkowe pole tekstowe
  - password – jak *text*, ale wpisywane znaki są maskowane
  - checkbox – pole wyboru (opcje niezależne od siebie)
  - radio – przycisk radiowy (opcje nawzajem wykluczające się)
  - submit – przycisk wysyłania formularza do serwera
  - reset – przycisk przywracający domyślne wartości kontrolek
  - file – kontrolka wyboru pliku (pole edycyjne + przycisk)
  - hidden – pole ukryte
  - image – graficzny odpowiednik *submit*
  - button – przycisk skryptu wykonywanego przez przeglądarkę

- Znaczenie pozostałych atrybutów częściowo zależy od typu kontrolki
- HTML5 wprowadził wiele specjalnych kontrolek tekstowych (email, date, time, …) – implementacja zależna od przeglądarki

Pole danych: pozostałe atrybuty

- name – dla kontrolek, które wysyłają dane do serwera (text, password, submit, checkbox, radio, file, hidden) – określa nazwę, pod jaką zawartość kontrolki (*value*) zostanie wysłana. Brak atrybutu name powoduje, że zawartość kontrolki w ogóle nie jest wysłana;
- value – znaczenie zależne od typu kontrolki:
  - text, password – wartość początkowa wyświetlana w kontrolce (przywracana po kliknięciu przycisku reset)
  - checkbox, radio – wartość wysłana o ile kontrolka jest zaznaczona (prze braku value wysyłana jest wart. "on"); jeżeli kontrolka nie jest zaznaczona, nic nie jest wysyłane
  - hidden – wartość niewidoczna, wysyłana do serwera
  - submit, reset, button – opis wyświetlany na przycisku
- checked – tylko kontrolki typu checkbox oraz radio – jeżeli występuje, oznacza, że początkowo kontrolka jest zaznaczona;
  - checkbox – każdy może być zaznaczony lub nie, niezależnie od pozostałych
  - radio – powinien być zaznaczony dokładnie jeden z każdej grupy przycisków (grupy są określone przez atr. name)
- disabled – jeżeli występuje, oznacza że kontrolka jest niedostępna dla użytkownika; stan ten można zmienić za pomocą skryptu wykonywanego po stronie przeglądarki; Zawartość kontrolki niedostępnej nie jest przesyłana na serwer
- readonly – jeżeli występuje, oznacza że kontrolka nie może być zmieniana przez użytkownika; Zawartość kontrolki jest przesyłana na serwer

- required – jeżeli występuje, to wypełnienie kontrolki jest wymagane, bez tego przeglądarka nie wyśle formularza
- size – tylko kontrolki typu text, password, file – szerokość pola przeznaczonego na wpisanie tekstu (wyrażona w znakach)
- maxlength – tylko kontrolki typu text, password – maksymalna ilość znaków, jakie można wpisać; Może mieć wartość większą, niż size – wówczas tekst będzie przewijany podczas wpisywania
- tabindex – kolejność aktywacji klawiszem tabulatora; domyślnie odpowiada kolejności występowania kontrolek w dokumencie;
- accesskey – klawisz szybkiego dostępu (skrót klawiaturowy);
  sposób aktywacji zależy od platformy – np. w Windows trzeba użyć kombinacji [Alt]+skrót;
  działanie jest zależne od typu kontrolki:
  - text, password, file – aktywacja
  - checkbox, radio – zmiana zaznaczenia
  Znaczenie accesskey jest ograniczone ze względu na trudność we wskazaniu skrótu (podkreślenie może być mylone z łączem)

Etykieta: label

- Reprezentuje etykietę przypisaną do pola danych formularza:

```
<input type="…" id="ID" name="…">
<label for="ID">Opis komponentu</label>
```

- Znacznik zamykający – wymagany
- Atrybuty:
  id, class, style, title,
  for – id elementu opisywanego przez etykietę
  accesskey – klawisz szybkiego dostępu (skrót klawiaturowy)
  onfocus – skrypt wykonywany po uzyskaniu "fokusu"
  onblur – skrypt wykonywany po utracie "fokusu"
- Działanie: aktywacja etykiety (skrót klawiaturowy lub kliknięcie) jest równoważne aktywacji pola danych;
  W przypadku kontrolek typu checkbox i radio – obowiązkowe!

Pole edycyjne: input type="text"

- Jednolinijkowe pole edycyjne do wpisania tekstu:

```
<input type="text" name="t1"
  value="tu wpisz…" size="25" maxlength="30" >
```

- Ważniejsze atrybuty:
  id – identyfikator, kojarzący pole z etykietą,
  name – nazwa, pod jaką zawartość będzie wysłana do serwera
  value – początkowa wartość
  size – rozmiar pola
  maxlength – maksymalna długość wpisanego tekstu

Pole edycyjne hasła: input type="password"

- Jednolinijkowe pole edycyjne do wpisania hasła:

```
<input type="password" name="p1"
  size="25" maxlength="30" >
```

- Ważniejsze atrybuty:
  id – identyfikator, kojarzący pole z etykietą,
  name – nazwa, pod jaką zawartość będzie wysłana do serwera
  size – rozmiar pola
  maxlength – maksymalna długość wpisanego tekstu

Pole wyboru: input type="checkbox"

- Pole edycyjne opcji niezależnych od siebie:

```
<input type="checkbox" id="ch1" name="ch1"
  value="1" checked >
<label for="ch1">Opis opcji 1</label>
<input type="checkbox" id="ch2" name="ch2"
  value="1">
<label for="ch2">Opis opcji 2</label>
```

- Ważniejsze atrybuty:
  id – identyfikator, kojarzący pole z etykietą,
  name – nazwa, pod jaką zawartość będzie wysłana do serwera
  value – wartość wysyłana, jeżeli pole jest zaznaczone
  (domyślnie "on"; jeżeli nie zaznaczone, nic nie jest wysyłane)
  checked – jeżeli wystąpi, pole jest zaznaczone

Przycisk radiowy: input type="radio"

- Pole edycyjne opcji niezależnych od siebie:

```
<input type="radio" id="r1" name="rg1"
  value="1" checked >
<label for="r1">Opis opcji 1</label>
<input type="radio" id="r2" name="rg1"
  value="2">
<label for="r2">Opis opcji 2</label>
```

- Ważniejsze atrybuty:
  id – identyfikator, kojarzący pole z etykietą,
  name – nazwa, pod jaką zawartość będzie wysłana do serwera;
  ważne! atrybut name dla grupy opcji musi być ten sam
  value – wartość wysyłana, jeżeli pole jest zaznaczone
  checked – jeżeli wystąpi, pole jest zaznaczone

Pole wysyłania pliku: input type="file"

- Pole edycyjne i przycisk wyboru pliku:

```
<input type="file" name="f1" size="25">
```

- Ważniejsze atrybuty:
  id – identyfikator, kojarzący pole z etykietą,
  name – nazwa, pod jaką zawartość będzie wysłana do serwera
  size – rozmiar pola
- W zwykły sposób (para name - value) wysyłana jest nazwa pliku, sam plik jako załącznik żądania HTTP; Przesyłanie pliku jest możliwe tylko metodą POST

Przycisk submit: input type="submit"

- Przycisk wysyłania formularza:

```
<input type="submit" name="s1" value="Wyślij">
```

- Ważniejsze atrybuty:
  value – napis wyświetlany na przycisku w przeglądarce
  name – jeżeli jest, to jest też wysyłana na serwer – można utworzyć kilka przycisków i sprawdzić który został użyty

Przycisk reset: input type="reset"

- Przycisk czyszczenia formularza:

```
<input type="reset" value="Wyczyść">
```

- Ważniejsze atrybuty:
  value – napis wyświetlany na przycisku w przeglądarce

Pole tekstowe: textarea

- Tworzy wielolinijkowe pole tekstowe:
  `<textarea rows="…" cols="…" name="…"> … </textarea>`
- Znacznik zamykający – obowiązkowy
- Atrybuty:
  id, class, style, title,
  name – nazwa, pod jaką zawartość będzie wysłana do serwera
  rows, cols – liczba widocznych wierszy i kolumn
  disabled – niedostępne dla użytkownika
  readolny – pole tylko do odczytu
  onfocus – skrypt wykonywany po uzyskaniu "fokusu"
  onblur – skrypt wykonywany po utracie "fokusu"
  onselect – skrypt wykonywany zmianie zaznaczenia tekstu
  onchange – skrypt wykonywany po modyfikacji zaznaczenia

Lista wyboru: struktura listy

```
<select>
 |-- <option> pozycja listy
 |-- <option> pozycja listy
 |-- <optgroup> nazwa grupy pozycji
 |     |-- <option>
 |     |-- <option>
 |-- <optgroup>
       |-- <option>
       |-- <option>
```

Lista wyboru może być wyświetlana jako lista zwykła lub lista rozwijana. Opcjonalne elementy *optgroup* tworzą hierarchiczne menu (jednak nie mogą być zagnieżdżone); nadają nazwę grupom pozycji listy, lecz same nie mogą być wybierane.

Lista wyboru: select

- Tworzy zwykłą lub rozwijaną listę wyboru:

```
<select size="…" name="…">
  …
</select>
```

- Znacznik zamykający – obowiązkowy
- Atrybuty:
  id, class, style, title,
  name – nazwa, pod jaką zawartość będzie wysłana do serwera
  size – rozmiar pola (znaczenie zależnie od typu)
  multiple – możliwość wyboru wielu pozycji na liście
  disabled – niedostępne dla użytkownika
  tabindex – kolejność aktywacji klawiszem tabulatora
  onfocus – skrypt wykonywany po uzyskaniu "fokusu"
  onblur – skrypt wykonywany po utracie "fokusu"
  onchange – skrypt wykonywany po modyfikacji zaznaczenia

Atrybuty

- name – nazwa, pod jaką zawartość kontrolki zostanie wysłana do serwera (wysyłana jest wartość określona przez atrybut *value* wybranego elementu *option*)
- multiple – jeżeli występuje, oznacza możliwość zaznaczenia więcej niż jednej pozycji na liście (np. w systemie Windows trzeba klikając wciskać klawisz [CTRL]). Lista wielokrotnego wyboru jest zawsze wyświetlana jako zwykła.
- size – określa sposób wyświetlania i rozmiar listy;
  Jeżeli nie występuje atrybut *multiple* i rozmiar ma wartość 1 (wartość domyślna), to wyświetlana jest lista rozwijana;
  Z kolei jeżeli rozmiar jest większy niż 1, to wyświetlana jest lista zwykła, zaś atrybut *size* określa liczbę widocznych na raz linijek

Element listy: option

- Reprezentuje pozycję na liscie wyboru:

```
<option value="…" selected>…</option>
```

- Znacznik zamykający – opcjonalny
- Atrybuty:
  id, class, style, title,
  value – wartość wysłana do serwera, o ile pozycja jest wybrana
  selected – pozycja początkowo oznaczona jako wybrana
  disabled – pozycja listy niedostępne dla użytkownika

Grupa elementów listy: optgroup

- Umożliwia tworzenie hierarchicznego menu:

```
<optgroup label="…">
  <option value="…" selected>…
</optgroup>
```

- Znacznik zamykający – wymagany
- Atrybuty:
  id, class, style, title,
  label – etykieta dla grupy opcji
  disabled – grupa niedostępna dla użytkownika

Lista wyboru: przykład 1

- Lista rozwijana

```
<select size="1" name="li1">
  <option value="1" selected>opcja 1
  <option value="2">opcja 2
  <option value="3">opcja 3
  <option value="4">opcja 4
 </select>
```

Lista wyboru: przykład 2

- Lista przewijana, z układem hierarchicznym

```
<select size="6" name="li2">
  <optgroup label ="grupa 1">
    <option value="1a" selected>opcja 1a
  </optgroup>
  <optgroup label ="grupa 2">
    <option value="2a">opcja 2a
  </optgroup>
 </select>
```

Formularze

Sposób przesyłania danych formularza do serwera zależy od wybranej metody HTTP:

- Metoda GET

```
<form method="GET" action="plik.php"
 enctype="application/x-www-form-urlencoded">
  …
</form>
```

- Metoda POST

```
<form method="POST" action="plik.php"
 enctype="multipart/form-data">
  …
</form>
```

Metoda GET

- Działanie
  dane z formularza są przesyłane w żądaniu HTTP:

```
GET /action.php?name=value&name=value HTTP/1.1
Host: www.sth.com
```

- Cechy:
  – dane żądania są widoczne i zapisywane w historii/ulubionych
  – spacje i znaki spoza ASCII są kodowane ("+", "%HH")
  – nie można przesyłać plików na serwer

Metoda POST

- Działanie
  dane z formularza są przesyłane w nagłówkach HTTP:

```
POST /action.php HTTP/1.1
Host: www.sth.com
name=value&name=value
```

- Cechy:
  – dane żądania nie są widoczne ani zapisywane w historii
  (ale dane nie są szyfrowane; bezpieczeństwo -> HTTPS)
  – można przesyłać pliki na serwer

Django

Generowanie strony WWW przez Django (streszczenie):

- Django odbiera dane z WSGI i tworzy obiekt HttpRequest;
  Dane z odebrane WSGI są przekształcane, m.in. tworzone są oddzielne struktury na dane z formularzy (w obiekcie request typu HttpRequest są to request.GET oraz request.POST)
- Django "dekoduje" URL żądania i przeszukuje pliki urls.py projektu oraz aplikacji – na tej podstawie wybiera widok
- Funkcja widoku tworzy potrzebny model, zgodnie z żądaniem oraz ewentualnie danymi z formularzy. Model aktualizuje swój stan (jeżeli to potrzebne) i dostarcza potrzebne dane;
- Funkcja widoku tworzy kontekst – zbiór wszystkich danych potrzebnych do wygenerowania strony WWW i używając szablonu renderuje dokument HTML; Następnie tworzy obiekt HttpResponse, który zwraca jako swój rezultat;

Formularze w Django

- Formularz można dodać do szablonu HTML – dane z formularza zostaną przeniesione do odpowiednich struktur HttpRequest; Funkcja widoku ma dostęp do danych z formularzy – można je dowolnie wykorzystać, są też w Django funkcje walidacji danych różnego typu – jednak wymaga to wiele zachodu…
- Formularz może być też wygenerowany przez Django, przez obiekt klasy formularza (Form lub ModelForm);
- ORM Django jest spójny z systemem formularzy Django: klasa ModelForm potrafi przenieść dane z modelu do formularza i potem z formularza do modelu, a nawet zapisać zmieniony model w bazie danych

- Django znacznie upraszcza obsługę formularzy, zapewniając automatyzację w zakresie wszystkich trzech etapów przetwarzania formularzy:
  – pobranie danych z modeli
  – tworzenie formularzy w dokumencie HTML
  – pobieranie i walidacja danych z formularzy

- Do obsługi formularzy są dwie klasy:
  – Form
  – ModelForm

Struktura aplikacji (nadal bardzo prostej) Django:

```
<aplikacja>
 - <static>
 - urls.py
 - views.py
 - models.py
 - forms.py
 - <templates>
    - szablon.html
```

- Forms.py – plik klas formularzy

Klasa Form

- Formularze zawierające arbitralnie wybrane kontrolki HTML, definiowane podobnie jak modele
- Zapewnia walidację danych na podstawie typu i opcji pól (np. dopuszczalny brak wartości, maksymalna długość, formalna poprawność adresu email, daty, …)
- Mogą służyć do pobierania danych, które nie są zapisywane w modelach, np.:
  – formularz wyszukiwania
  – formularz kontaktowy (jeżeli dane są wysyłane emailem)
  albo pobierania/edycji danych w nietypowych konfiguracjach, np. pochodzących z kilku różnych modeli

Klasa ModelForm

- Formularze tworzone na podstawie modeli - automatycznie wyposażane w pola do wprowadzania/edycji wszystkich danych modelu, z uwzględnieniem typu danych oraz niektórych ograniczeń (np. dopuszczalny brak wartości, maksymalna długość, formalna poprawność adresu email, daty, …)
- Wyposażone w metody do zapisania danych w modelu (całkowicie automatycznie albo w sposób kontrolowany)
- Można dodać dodatkowe pola (ale oczywiście trzeba we własnym zakresie zadbać o ich wykorzystanie)
- Można zmienić domyślne działanie/opcje przez klasę Meta
- Można dodać własne mechanizmy walidacji danych
- Klasy pól formularzy są odpowiednikami klas pól modeli, w większości o tej samej nazwie (np. EmailField, DateField, FileField, …), z kilkoma wyjątkami:
  – nie ma pól AutoField, BigAutoField, SmallAutoField
  (nie mają sensu – wartość dostarcza sama baza danych)
  – CharField jest odpowiednikiem CharField i TextField modeli,
  chociaż zależnie od opcji może być reprezentowane przez
  różne kontrolki formularza HTML : input type='text'
  (domyślnie dla CharField) albo textarea (domyślnie TextField)
  – ModelChoiceField reprezentuje ForeignKey modeli
  – ModelMultipleChoiceField reprezentuje ManyToManyField
- Dokładnie te same klasy pól są dla klas Form i ModelForm;
  ModelForm jest wygodniejsza, bo "przejmuje" pola z modelu
- Pola formularza domyślnie zachowują niektóre ograniczenia i opcje pól modeli, np.:
  – opcja max_length jest przenoszona do HTML przez atrybut
  maxlength kontrolki input
  – opcja blank pola modelu determinuje atrybut required HTML:
  jeżeli blank=False, to jest dodawany atrybut required,
  a w efekcie formularza nie da się wysłać bez wypełnienia pola
  – opcja verbose_name jest przenoszona do HTML jako
  treść etykiety (label) formularza
- Wszystkie opcje można zmienić przez klasę Meta formularza
  albo "nadpisanie" pól przejętych z modelu

Klasa ModelForm (cd.)

- Musi dziedziczyć po klasie django.forms.ModelForm i wskazywać którego modelu dotyczy
- Może zawierać dodatkowe lub nadpisane pola oraz opcje

```
from django import forms
from .models import Post

class PostForm(forms.ModelForm):

    class Meta:
        model = Post
```

Klasa Meta klasy ModelForm

- Musi wskazać z którego modelu formularz przejmuje pola:

```
model = <nazwa klasy modelu>
```

- Może zawierać opcje:
  – exclude – nazwy pól modelu, które zostaną pominięte w formularzu (pozostałe będą w tej kolejności, co w modelu)
  – fields – nazwy pól modelu, które maja być uwzględnione w formularzu, z określeniem ich kolejności;
  Nalży użyć albo exclude albo fields, w formie krotki
  – widgets – słownik przyporządkowujący polom formularza typy kontrolek HTML; umożliwia podanie atrybutów HTML (co nie jest możliwe przez klasy pól formularzy),
  np. Django nie korzysta z atrybutu placeholder…
  – labels – słownik przyporządkowujący polom formularza treść opisów kontrolek HTML (w elementach label HTML)
- Przykład klasy Meta dla formularza modelu Post:

```
class Meta:
  model = Post
  fields = ('title', 'text')
  labels = {
    'title': 'Tytuł wpisu',
    'text': 'Treść wpisu',
  }
  widgets = {
    'title': TextInput(
      attrs={'placeholder': 'Tu wpisz tytuł'}
    )
  }
```

Generowanie formularza HTML w szablonie Django

- Obiekt formularza, zarówno klasy Form, jak i ModelForm,
  jest w szablonie automatycznie konwertowany na HTML
- Do wyboru są 3 metody:
  - as_p – kontrolki są umieszczane w oddzielnych akapitach, najpierw element label, potem kontrolka input/textarea/selekt
  - as_ul – kontrolki są umieszczane w elementach listy, jw.
  - as_table – kontrolki są umieszczane w wierszach tabeli,
    najpierw etykieta (w komórce th), potem kontrolka (td)
- Żadna z metod nie dodaje do HTML elementu form ani przycisku wysyłania formularza (np. button type='submit)
- Metody as_ul oraz as_table nie dodają do HTML początku oraz końca listy (tabeli)
- Przykład generowania formularza:

```
<form method="POST" >{% csrf_token %}
  {{ form.as_p }}
  <p><button type="submit">Zapisz</button></p>
</form>
```

- Znacznik Django csrf_token jest niezbędny, jeżeli jest włączony middleware ochrony przed atakami CSRF (XSRF), Cross Site Request Forgery
- Imponująco trywialne – jest tylko jedna drobna wada: bez formatowania w CSS formularz nie wygląda zbyt dobrze…
- Typowy sposób generowania formularza ma dwie szczególne cechy, na które należy zwrócić uwagę:
  - Pominięcie atrybutu action (albo równoważnie action="") powoduje przesłanie formularza pod ten adres URL, z którego formularz został pobrany – w Django trafi do tej samej funkcji widoku, która go utworzyła
  - Przesyłanie metodą POST pozwala łatwo poznać czy formularz jest już wypełniony – obiekt HttpResponse w funkcji widoku zawiera rodzaj żądania (request.method) o wartości "POST"

Stany formularzy

Django definiuje dwa stany formularzy:

- Formularze niezwiązane (unbound)
  nie mają przypisanych danych – kiedy zostaną przekształcone na HTML, będą puste
- Formularze związane (bound)
  mają przypisane dane (np. wypełnione przez użytkownika), wobec czego mogą być walidowane; Walidacja polega na sprawdzeniu czy każde pole zawiera poprawne (valid) dane; Formularz związany, który się nie waliduje, przekształcony na HTML powinien zawierać informacje o błędach – przy polach, których wartości nie są poprawne (plus ewentualnie stosowne formatowanie takich pól za pomocą CSS)
- Django zawiera mechanizmy automatycznej walidacji (na podstawie opcji pól modelu) i wyświetlania błędów w HTML

Użycie formularza w funkcji widoku:

- Typowo w Django ten sam widok obsługuje wszystkie etapy "życia" formularza, np. dla formularza dodawania wpisu:
    - Utworzenie pustego (niezwiązanego) formularza i wyświetlenie go na stronie
    - Walidacja danych wypełnionego (związanego) formularza
    - Ponowne wyświetlenie go na stronie jeżeli zawiera błędy
    - Zapisanie danych w bazie danych
- Bardzo podobnie będzie przebiegała edycja wpisu – tyle tylko, że formularz zostanie od razu wypełniony danymi z modelu
- Funkcja widoku jest "wieloprzebiegowa" – realizuje różne zadania zależnie od etapu "życia" formularza i danych

Postępowanie przy tworzeniu nowego wpisu:

```
from django.shortcuts import render, redirect
from .models import Post, Comment
from .forms import PostForm

def post_new(request):
  if request.method == "POST":
    form = PostForm(request.POST)
    if form.is_valid():
      post = form.save(commit=False)
      post.author = request.user
      post.save()
      return redirect('post', id=post.id)
  else:
    form = PostForm()
  return render(request, 'post_new.html', {'form': form})
```

- Jeżeli request.method != "POST", to widok jest wyświetlany pierwszy raz i tworzony jest pusty formularz
- Jeżeli request.method == "POST", to formularz HTML jest już wypełniony i następuje przeniesienie danych do obiektu formularza Django i jego walidacja
- Jeżeli formularz się waliduje, to dane zostają zapisane do obiektu modelu; Funkcja save z argumentem commit=False,
  post = form.save(commit=False)
  nie zapisuje danych od razu do bazy danych, można zatem wykonać dodatkowe operacje – np. uzupełnić dane modelu – oraz zapisać model instrukcją post.save(); Przekierowanie powoduje wyświetlenie strony pokazującej efekt końcowy
- Jeżeli formularz się nie waliduje, to nie nastąpi jego zapisanie; Walidacja doda informacje o błędach do formularza, a formularz zostanie wyświetlony ponownie

Postępowanie przy edycji istniejącego wpisu:

```
from django.shortcuts import render, redirect
from .models import Post, Comment
from .forms import PostForm

def post_edit(request, id):
  post = Post.objects.get(id=id)
  if request.method == "POST":
    form = PostForm(request.POST, instance=post)
    if form.is_valid():
      post = form.save(commit=False)
      post.published_date = timezone.now()
      post.save()
      return redirect('post', id=post.pk)
  else:
    form = PostForm(instance=post)
  return render(request, 'post_new.html', {'form': form})
```

- Na początek należy pobrać dane, które maja być edytowane:
  post = Post.objects.get(id=id)
- Schemat dalszego postępowania jest bardzo podobny jak przy dodawaniu nowego wpisu, jednak przy tworzeniu obiektu formularza Django należy podać jako dodatkowy argument obiekt modelu, który ma być zmodyfikowany;
  Przy pierwszym wyświetleniu formularza:
  form = PostForm(instance=post)
  natomiast po zmianach dokonanych przez użytkownika:
  form = PostForm(request.POST, instance=post)
- Pozostałe czynności są identyczne, łącznie z odroczeniem zapisu do bazy danych: post = form.save(commit=False), zapisaniem ich po wykonaniu dodatkowych operacji: post.save() oraz przekierowaniem

Walidacja formularzy

Walidacja formularzy może być realizowana na dwa sposoby:

- Po stronie przeglądarki
  – sama przeglądarka dokonuje podstawowej walidacji
  (np. atrybut required, nowe typy kontrolek – date, email, …),
  – można również dołączyć skrypt Javascript/jQuery…
  Użytkownik może łatwo obejść te ograniczenia (także celowo!). Ta walidacja jest dla użytkownika – ogranicza przeładowania strony przy nieprawidłowym wypełnieniu formularza
- Po stronie serwera
  Użytkownik nie może jej wyłączyć – jedyna, która zapewnia poprawne działanie serwisu – o ile jest zrobiona prawidłowo; Jest też elementem zabezpieczeń przed atakami, np. CSRF

Walidacja formularzy w Django

- Django dokonuje automatycznej walidacji danych formularzy, na podstawie opcji modelu i formularza (w tym klas Meta); Niestety rzadko kiedy to wystarcza. Typowy przykład: rejestracja użytkownika i wymóg unikalnego adresu email
- Dodatkowa walidacja danych może być zrealizowana na kilka sposobów:
    - Wskazanie dodatkowych walidatorów w opcjach pola modelu lub formularza (wbudowane walidatory Django lub własne)
    - Zdefiniowanie własnej podklasy pola modelu i jego walidacji
    - Nadpisanie w klasie formularza funkcji:
      `clean_<pole>()` – walidacja poszczególnych pól formularza,
      `clean()` – walidacja formularza jako całości (np. wzajemnych relacji pomiędzy wartościami pól, kiedy już wiadomo, że pola zawierają wartości same w sobie poprawne)

Funkcja walidacji pola

- Funkcja walidacji pola formularza powinna:
  – wywołać odziedziczoną funkcję walidacji
  – pobrać wstępnie "oczyszczoną" wartość pola z formularza
  – dodać opis błędu do pola, jeżeli wartość jest nieprawidłowa
  – zwrócić jako rezultat wartość "oczyszczoną" pola

```
def clean_title(self):
    super().clean()
    title = self.cleaned_data["title"]
    if <warunek wystąpienia błędu>:
        self.add_error("title", "<opis błędu>")
    return title
```

Funkcja walidacji formularza

- Funkcja walidacji całości formularza powinna:
  – wywołać odziedziczoną funkcję walidacji
  – pobrać oczyszczone wartość pól z formularza
  – dodać opisy błędów do pól, jeżeli wartości są nieprawidłowe, np. sprzeczne

```
def clean(self):
    super().clean()
    title = self.cleaned_data["title"]
    text = self.cleaned_data["text"]
    if <warunek wystąpienia błędu>:
        self.add_error("title", "<opis błędu>")
        self.add_error("text", "<opis błędu>")
```

Wyświetlanie informacji o błędach w formularzu HTML

- Do formularza z błędami dodawane są informacje o tych błędach, w postaci listy <ul> z klasą CSS "errorlist" (niezależnie od metody generowania formularza). Każdy opis błędu dodany funkcją add_error jest osobnym elementem tej listy, np.:

```
<ul class="errorlist">
  <li>To pole jest wymagane.</li>
  <li>Tytuł nie może zawierać samych spacji.</li>
</ul>
<p>
  <label for="id_title">Tytuł:</label>
  <input type="text" name="title" value=" " [itd.] >
</p>
```

- Właściwe formatowanie błędów można uzyskać w CSS stosując odpowiednio relacje selektorów, np.: lista: ul.errorlist, element listy: ul.errorlist li, etykieta kontrolki: ul.errorlist + p label, kontrolka: ul.errorlist + p input itp.

Przykład formatowania formularzy as_p() Django w CSS

```
input, textarea, select {
  font: 500 normal 12pt Dosis;
  color: gray;
  padding: 6px 12px;
  width: 500px;
  border: 1px solid lightsteelblue;
  border-radius: 5px;
  outline: none;
}

textarea {
  resize:vertical;
}

input:focus, textarea:focus, select:focus {
  color: dimgray;
  border: 1px solid steelblue;
  outline:none;
}

form button {
  font: 800 normal 14pt Dosis;
  padding: 6px 12px;
  width: 500px;
  border: 1px solid lightsteelblue;
  background-color: white;
  color: steelblue;
  border-radius: 5px;
  outline: none;
}

button:hover, button:focus {
  background-color: steelblue;
  border: 1px solid steelblue;
  color: white;
}

form p {
  display: grid;
  grid-template-columns: 200px 500px;
  margin: 10px 0px;
  padding: 0;
}

form label {
  font: normal 14pt Raleway;
  color: steelblue;
}

ul.errorlist {
  display: block;
  margin: 0px;
  padding: 0px;
}
```

```
ul.errorlist li {
  display: block;
  margin: 0px;
  padding: 0px 0px 0px 200px;
  font: 500 normal 12pt Dosis;
  color: red;
}

ul.errorlist + p {
  margin-top: 3px;
}

ul.errorlist + p label {
  color: red;
}

ul.errorlist + p input,
ul.errorlist + p textarea,
ul.errorlist + p select {
  border: 1px solid red;
}

ul.errorlist + p input:focus,
ul.errorlist + p textarea:focus,
ul.errorlist + p form select:focus {
  box-shadow: 1px 1px 2px lightcoral;
}

@media (max-width: 950px) {
  form p {
    display: grid;
    grid-template-columns: 200px auto;
    margin: 10px 0px;
  }

  input, textarea, select, button {
    width: 100%;
  }
}

@media (max-width: 700px) {
  form p {
    display: grid;
    grid-template-columns: auto;
    margin: 10px 0px;
  }

  input, textarea, select, button {
    width: 100%;
  }

  ul.errorlist li {
    padding: 0px;
  }
}
```

Zadania

Proszę uzupełnić projekt Django z poprzedniego ćwiczenia o formularze do dodawania wpisów i komentarzy, a w tym celu:

1. Utworzyć plik forms.py z definicją klasy formularza wpisu; Formularz powinien zawierać tylko pola `title` i `text`, pozostałe informacje zostaną dodane w funkcji widoku
2. Do szablonu aplikacji dodać odnośnik do strony dodawania wpisu
3. Dodać funkcję widoku do obsługi formularza oraz szablon wyświetlania strony dodawania wpisu
4. Dodać funkcję widoku edycji wpisu; Widok edycji wpisu wykorzystuje tę samą klasę formularza, co funkcja widoku dodawania wpisu, może też wykorzystywać ten sam szablon – ale można dodać nowy, dedykowany dla edycji wpisu
5. Dodać zabezpieczenie, dzięki któremu dodawanie i edycja wpisów będzie dostępne tylko dla zalogowanych użytkowników
6. Do arkusza stylów dodać formatowanie formularzy (można posłużyć się przykładem powyżej, należy tylko dostosować czcionki, kolory i szerokości poszczególnych elementów)

   Przy realizacji punktów 1-5 proszę skorzystać z poradnika "Django Girls": https://tutorial.djangogirls.org/pl/django_forms/

7. Do pliku forms.py dodać definicję klasy formularza komentarza; Formularz powinien zawierać tylko pole `text`, pozostałe informacje zostaną dodane w funkcji widoku
8. Do szablonu używanego do wyświetlania wpisu dodać odnośnik do strony dodawania komentarza; Dodać funkcję widoku dodawania komentarza oraz odpowiedni szablon; W funkcji widoku należy zadbać o to, aby wraz z treścią komentarza została również zapisana informacja o jego autorze (analogicznie, jak przy wpisach) oraz o wpisie, do którego komentarz się odnosi. Dodać zabezpieczenie, dzięki któremu dodawanie komentarzy będzie dostępne tylko dla zalogowanych użytkowników
9. * Rozszerzyć zabezpieczenie serwisu z punktu 5, aby edycji wpisu mógł dokonać tylko jego autor, a nie dowolny zalogowany użytkownik;
10. ** Do definicji klas formularzy z punktów 1 i 7 dodać własne funkcje walidacji poszczególnych pól formularzy, z arbitralnie wybranymi regułami walidacji (np. za błąd można uznać wystąpienie określonych znaków lub słów w tytule lub treści, zbyt dużą lub zbyt małą liczbę znaków lub słów w tytule, powtórzenie tytułu istniejącego wpisu itp.)