Laboratorium 3

1. Identyfikacja błędu - shell

```
>>> import datetime
>>> from django.utils import timezone
>>> from polls.models import Question
# utworzenie instancji Question z pub_date z data przesuniętą 30 dni
w przyszłość
>>> future_question = Question(pub_date=timezone.now() +
datetime.timedelta(days=30))
# co zostało ostatnio opublikowane?
>>> future_question.was_published_recently()
True
```

2. Tworzenie testu aby znaleźć błąd
   nazwa_aplikacji/tests.py

```
import datetime

from django.utils import timezone

from django.test import TestCase

from .models import Question

class QuestionMethodTests(TestCase):



    def test_was_published_recently_with_future_question(self):
        """
        was_published_recently() should return False for questions whose
        pub_date is in the future.
        """
        time = timezone.now() + datetime.timedelta(days=30)
        future_question = Question(pub_date=time)
        self.assertIs(future_question.was_published_recently(), False)
```

- uruchamianie testów

```
python manage.py test nazwa_aplikacji
```

3. Naprawianie błędów

   nazwa_aplikacji/models.py

```
def was_published_recently(self):
    now = timezone.now()
    return now - datetime.timedelta(days=1) <= self.pub_date <= now
```

4. Bardziej kompleksowe testy
   - nazwa_aplikacji/tests.py

```
def test_was_published_recently_with_old_question(self):
    """
    was_published_recently() should return False for questions whose
    pub_date is older than 1 day.
    """
    time = timezone.now() - datetime.timedelta(days=30)
    old_question = Question(pub_date=time)
    self.assertIs(old_question.was_published_recently(), False)


def test_was_published_recently_with_recent_question(self):
    """
    was_published_recently() should return True for questions whose
    pub_date is within the last day.
    """
    time = timezone.now() - datetime.timedelta(hours=1)
    recent_question = Question(pub_date=time)
    self.assertIs(recent_question.was_published_recently(), True)
```

5. Testowanie widoku – test klienta
- shell

```
>>> from django.test.utils import setup_test_environment
>>> setup_test_environment()
>>> from django.test import Client
# utworzyć instancję klienta dla naszego użytku
>>> client = Client()
```

- mając to gotowe, możemy zwrócić się do klienta, aby popracował dla nas

```
# uzyskanie odpowiedź od '/'
>>> response = client.get('/')
>>> response.status_code
```

```
404

# Z drugiej strony, aby znaleźć coś, co '/ aplikacja / ' użyjemy
'reverse()' zamiast zakodowanego na stałe URL

>>> from django.urls import reverse

>>> response = client.get(reverse('polls:index'))

>>> response.status_code

200

>>> response.content

b'\n    <ul>\n    \n        <li><a href="/polls/1/">What&#39;s
up?</a></li>\n    \n    </ul>\n\n'

>>> Jeśli nie działa, prawdopodobnie pominięto wezwanie do
setup_test_environment() opisany powyżej

>>> response.context['latest_question_list']

<QuerySet [<Question: What's up?>]>
```

6. Ulepszenie widoku

- nazwa_aplikacji/views.py

```
from django.utils import timezone

 class IndexView(generic.ListView):

     template_name = 'nazwa_aplikacji/index.html'

     context_object_name = 'latest_question_list'

     def get_queryset(self):

         """Return the last five published questions."""

         return Question.objects.order_by('-pub_date')[:5]


 def get_queryset(self):

     """

     Return the last five published questions (not including those set to
 be

     published in the future).

     """

     return Question.objects.filter(pub_date__lte=timezone.now()

 ).order_by('-pub_date')[:5]
```

7. Testowanie nowego widoku

- nazwa_aplikacji/tests.py

```
from django.urls import reverse

def create_question(question_text, days):
    """
    Creates a question with the given `question_text` and published the
    given number of `days` offset to now (negative for questions
published
    in the past, positive for questions that have yet to be published).
    """
    time = timezone.now() + datetime.timedelta(days=days)
    return Question.objects.create(question_text=question_text,
pub_date=time)



class QuestionViewTests(TestCase):
    def test_index_view_with_no_questions(self):
        """
        If no questions exist, an appropriate message should be
displayed.
        """
        response = self.client.get(reverse('polls:index'))
        self.assertEqual(response.status_code, 200)
        self.assertContains(response, "No polls are available.")

self.assertQuerysetEqual(response.context['latest_question_list'], [])


    def test_index_view_with_a_past_question(self):
        """
        Questions with a pub_date in the past should be displayed on the
        index page.
        """
```

```
        create_question(question_text="Past question.", days=-30)
        response = self.client.get(reverse('polls:index'))
        self.assertQuerysetEqual(
            response.context['latest_question_list'],
            ['<Question: Past question.>']
        )

    def test_index_view_with_a_future_question(self):
        """
        Questions with a pub_date in the future should not be displayed
on
        the index page.
        """
        create_question(question_text="Future question.", days=30)
        response = self.client.get(reverse('polls:index'))
        self.assertContains(response, "No polls are available.")
self.assertQuerysetEqual(response.context['latest_question_list'], [])

    def test_index_view_with_future_question_and_past_question(self):
        """
        Even if both past and future questions exist, only past questions
        should be displayed.
        """
        create_question(question_text="Past question.", days=-30)
        create_question(question_text="Future question.", days=30)
        response = self.client.get(reverse('polls:index'))
        self.assertQuerysetEqual(
            response.context['latest_question_list'],
            ['<Question: Past question.>']
        )
```

```
    def test_index_view_with_two_past_questions(self):
        """
        The questions index page may display multiple questions.
        """
        create_question(question_text="Past question 1.", days=-30)
        create_question(question_text="Past question 2.", days=-5)
        response = self.client.get(reverse('polls:index'))
        self.assertQuerysetEqual(
            response.context['latest_question_list'],
            ['<Question: Past question 2.>', '<Question: Past question
1.>']
        )
```

8. Testowanie DetailView()

- nazwa_aplikacji/views.py

```
class DetailView(generic.DetailView):
    ...
    def get_queryset(self):
        """
        Excludes any questions that aren't published yet.
        """
        return Question.objects.filter(pub_date__lte=timezone.now())
```

- nazwa_aplikacji/tests.py

```
    class QuestionIndexDetailTests(TestCase):
        def test_detail_view_with_a_future_question(self):
            """
            The detail view of a question with a pub_date in the future
    should
            return a 404 not found.
            """
```

```
        future_question = create_question(question_text='Future
    question.', days=5)

        url = reverse('polls:detail', args=(future_question.id,))

        response = self.client.get(url)

        self.assertEqual(response.status_code, 404)


    def test_detail_view_with_a_past_question(self):

        """

    The detail view of a question with a pub_date in the past should
    display the question's text.

        """

        past_question =
    create_question(question_text='PastQuestion.', days=-5)

        url = reverse('polls:detail', args=(past_question.id,))

        response = self.client.get(url)

    self.assertContains(response, past_question.question_text)
```

9. Wygląd

- nazwa_aplikacji/static/nazwa_aplikacji/style.css

```
li a {

    color: green;

}
```

- nazwa_aplikacji/templates/nazwa_aplikacji/index.html

```
{% load static %}


<link rel="stylesheet" type="text/css" href="{% static 'polls/style.css'
%}" />
```

10. Poprawienie wyglądu panelu administracyjnego

- nazwa_aplikacji/admin.py - zastąpić

```
from django.contrib import admin

from .models import Question

class QuestionAdmin(admin.ModelAdmin):
```

```
    fields = ['pub_date', 'question_text']


admin.site.register(Question, QuestionAdmin)
```

- nazwa_aplikacji/admin.py

```
from django.contrib import admin

from .models import Question

class QuestionAdmin(admin.ModelAdmin):

    fieldsets = [

        (None,               {'fields': ['question_text']}),

        ('Date information', {'fields': ['pub_date']}),

    ]

admin.site.register(Question, QuestionAdmin)
```

- nazwa_aplikacji/admin.py

```
from django.contrib import admin

from .models import Choice, Question
# ...
admin.site.register(Choice)
```

- nazwa_aplikacji/admin.py

```
from django.contrib import admin

from .models import Choice, Question

class ChoiceInline(admin.StackedInline):
    model = Choice
    extra = 3


class QuestionAdmin(admin.ModelAdmin):
    fieldsets = [
        (None,               {'fields': ['question_text']}),
        ('Date information', {'fields': ['pub_date'], 'classes':
['collapse']}),
    ]
    inlines = [ChoiceInline]

admin.site.register(Question, QuestionAdmin)
```

- nazwa_aplikacji/admin.py

```
class ChoiceInline(admin.TabularInline):
```

-nazwa_aplikacji/admin.py

```
class QuestionAdmin(admin.ModelAdmin):

    # ...

    list_display = ('question_text', 'pub_date',
'was_published_recently')
```

-nazwa_aplikacji/models.py

```
class Question(models.Model):

    # ...

    def was_published_recently(self):

        now = timezone.now()

        return now - datetime.timedelta(days=1) <= self.pub_date <= now

    was_published_recently.admin_order_field = 'pub_date'

    was_published_recently.boolean = True

    was_published_recently.short_description = 'Ostatnio opublikowane?'
```

- nazwa_aplikacji/admin.py – dodanie zmiennej w funkcji **QuestionAdmin**:

```
list_filter = ['pub_date']
```