Rich GUI with Django Packages

Maxim Mozgovoy

Django and GUI

- Django is not really frontend-oriented:
  its user interface capabilities are relatively limited.
- If a truly rich GUI is needed, use specialized frameworks (React / Angular / Vue…) Django can be used with Vue.js!
- Though for most simple interface elements Django is sufficient.
- Fortunately, Django is flexible enough to support numerous 3rd-party libraries and addons for improved user interface.
- Rule of thumb: if something is not easily done in Django, search for a 3rd-party solution for it.
- Start here: https://djangopackages.org/

Example: Bootstrap DatePicker+

- For many apps (including my Dentistry)
  a convenient calendar widget is required.
- Django provides a very simplistic calendar widget
  (based on a text field or a drop-down list).
- Solution: use the independent Bootstrap-datepicker-plus library!

Installing Bootstrap DatePicker+

- Run

```
python -m pip install django-bootstrap4
python -m pip install
                 django-bootstrap-datepicker-plus
```

- Add `'bootstrap4'` and `'bootstrap_datepicker_plus'` to the `INSTALLED_APPS` list inside `settings.py`.
- Insert this fragment into the `HEAD` section of the HTML page where you are going to use the widget:

```
{% load bootstrap4 %}
{% bootstrap_css %}
{% bootstrap_javascript jquery='full' %}
```

Using Bootstrap DatePicker+

- In your form specify `DatePickerInput` in the `widget` argument of a form field:

```
from bootstrap_datepicker_plus import
                                  DatePickerInput
…
timeslot = forms.DateField(widget =
                             DatePickerInput())
```

- Available widgets:

```
DateTime-Picker -- calendar and time picker
Date-Picker     -- calendar only
Time-Picker     -- time picker only
Month-Picker    -- month picker only
Year-Picker     -- year picker only
```


Bootstrap DatePicker+ Config

- More info at:
  https://github.com/monim67/django-bootstrap-datepicker-plus

- Pass to the widget the `options` argument.
  It is a dictionary of textual keys and values. Some nice options:

  `format`: format of the date (e.g., 'dd.mm.yyyy')
  `minDate`: the earliest selectable date
  `maxDate`: the latest selectable date
  `daysOfWeekDisabled`: list of disabled days of week
  (e.g., [0, 2, 6] = Sun, Tue, Sat)
  `showTodayButton`: display the "Today" button (True / False)

- Note: if you use the `format` argument, you will also have to set the `input_formats` argument of `DateField` similarly.

Displaying Bootstrap DatePicker+

To render a calendar widget inside *`our_form`* form:

- Insert `{{ our_form.media }}` sequence into HTML document's HEAD section.

- You can render the form as usual:
  `{{ our_form }}`

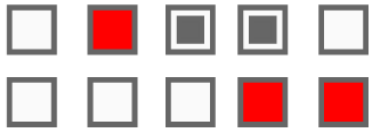
- However, date picker does not look good in this scheme.
  `{{ our_form.as_table }}`
  looks prettier.

- The command `as_table` wraps each widget on the form into `<tr>...</tr>` tags (so you have to use it inside `<table>`).

Dynamic Form Generation

- Sometimes you need to generate a form using database items.

- For example, in the movie theatre booking system vacant seats can be represented with checkboxes (so you can select them), while occupied seats will be just non-interactive elements:



- This can be achieved with dynamic form generation:
  the fields of the form can be customized according to data.

Dynamic Form Generation

```
class TestForm(forms.Form):
    def __init__(self, *args, **kwargs):
        vacant = kwargs.pop('vacant')
        super().__init__(*args, **kwargs)
        for i in range(0, len(vacant)):
            self.fields['seat_' + str(i)] =
                forms.BooleanField(label='')
                if vacant[i] else
                forms.CharField(widget =
                                forms.HiddenInput())
...
f = TestForm(vacant=[True, True, False, True])
```


Dynamic Form Generation

- Now we have a list of fields. Each field is either a “hidden input” or a checkbox. Note that fields array keeps the field in order of insertion (it is based on `OrderedDict` collection).
- Now we need to render this list field by field in HTML:

```
{% for field in dynamic_form %}
    {% if field.is_hidden %}
        [X]          <!-- just print [X] -->
    {% else %}
        {{ field }} <!-- render this field -->
    {% endif %}
{% endfor %}
```


Handling Binary Files and Images

- Django provides two types of database fields for handling binary data: `FileField` and `ImageField`.
- `ImageField` is an extended version of `FileField`: it can ensure that the stored file is actually an image.
- Physically these fields store only file metadata. The files themselves are located on the filesystem.
- Using these fields, your application can retrieve file URLs and access the files.

Example: Using ImageField

- To use `ImageField` fields, we must install `Pillow` package:
  `python -m pip install Pillow`
- Then we should specify the storage folder in `settings.py`:

```
BASE_DIR = … # insert after this line
MEDIA_ROOT = os.path.join(BASE_DIR, 'media')
MEDIA_URL = '/media/'
```

- Next, we should make media URL accessible from Django web server. Modify `urlpatterns` list in `urls.py` as follows:

```
from django.conf import settings
from django.conf.urls.static import static
…
urlpatterns = […] + static(settings.MEDIA_URL,
                           document_root=settings.MEDIA_ROOT)
```

Example: Using ImageField

- Let's modify our `Manufacturer` class to include an *optional* manufacturer logo:

```
class Manufacturer(models.Model):
  name = models.CharField(max_length=20)
  year_founded = models.IntegerField()
  logo = models.ImageField(upload_to='photos',
       null=True) # can be NULL in the database
```

- Let's create the corresponding input form:

```
class ManufacturerForm(forms.Form):
  name = forms.CharField(label='name',
                         max_length=20)
  year_founded =
           forms.IntegerField(label='year')
  logo = forms.ImageField(label='logo',
                          required=False)
```


Example: Using ImageField

To create a new `Manufacturer` with form data use:

```
f = ManufacturerForm(request.POST,
                     request.FILES)
if f.is_valid():
  logofile = (None
    if f.cleaned_data['logo'] is None
    else request.FILES['logo'])

  Manufacturer.objects.create(
     name=f.cleaned_data['name'],
     year_founded=f.cleaned_data['year_founded'],
     logo=logofile)
```


Example: Using ImageField

To display a `Manufacturer`'s logo use:

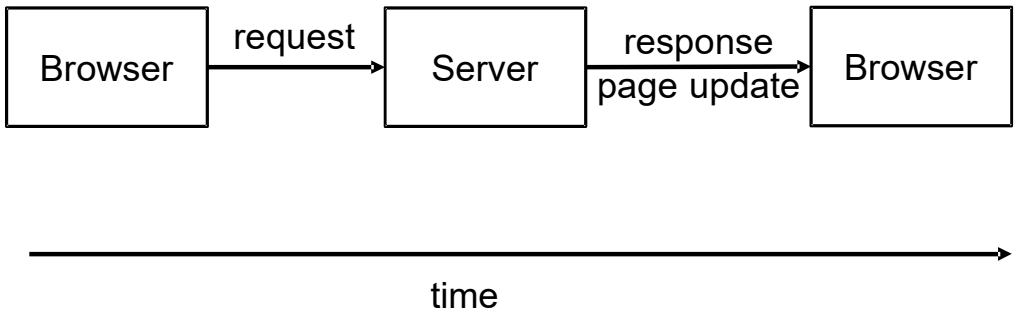
```
m = Manufacturer.objects.filter(name='BMW').get()

result = "no logo" if m.logo.name is None else
         "<img src=" + m.logo.url + ">"
```


Ajax

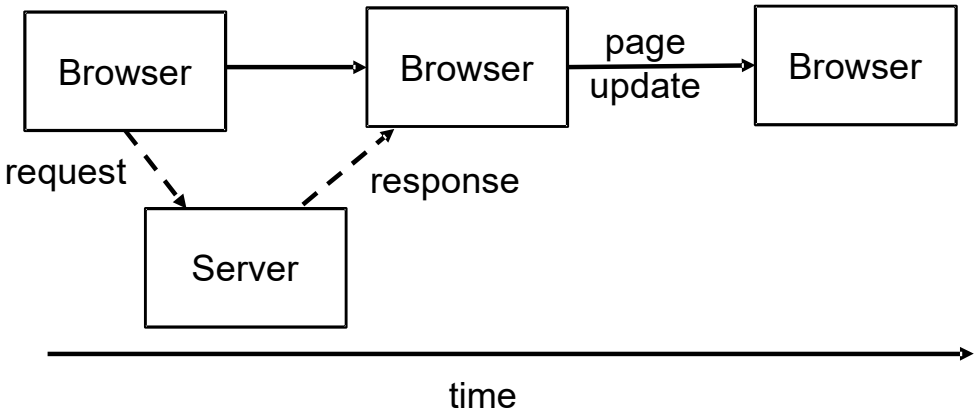
- Ajax (Asynchronous JavaScript and XML):
  the technology that enables web applications to establish *asynchronous* communication between client and server parts.

- Synchronous web communication:



Ajax

- Ajax (Asynchronous JavaScript and XML): the technology that enables web applications to establish *asynchronous* communication between client and server parts.
- Ajax web communication:



Ajax

- Ajax queries initiate server-side data processing, but the current page in the browser does not change.
- Server response triggers browser-side JavaScript code that has to process the response (& usually to modify the current page).
- Django has a handy helper `djangoajax` library. Install it:
  `python -m pip install djangoajax`
- Add `'django_ajax'` into the `INSTALLED_APPS` list and include this link into the HEAD section of your HTML file:

```
<script type="text/javascript"
src="https://cdn.jsdelivr.net/gh/yceruto/django-
ajax@2.3.7/django_ajax/static/django_ajax/js/jque
ry.ajax.min.js"></script>
```


Simple Ajax Example

- Ajax processing is implemented in ordinary Django views.
- Ajax view function should be decorated with `@ajax`.
- Use `get()` method of a request's `POST` object to access args.
- The result should be a dictionary of string key/value pairs.

```
from django_ajax.decorators import ajax
…
@ajax
def test_ajax(request): # calculate arg*arg
    n = int(request.POST.get('number'))
    return {'result': str(n*n) }

# let's assume this function is bound
# to the URL /test_ajax
```


Simple Ajax Example

- On the client side, there should be two JavaScript functions.
- The first initiates Ajax request, the second handles results.

```
<!-- inside HEAD section -->
<script type="text/javascript">
    function testAjax() {
        ajaxPost("/test_ajax", {'number': '5'},
            function (r) { alert(r.result); });
    }
</script>

…
<input type=button value="Test"
       onclick="testAjax();">
```

Simple Ajax Example

```
<script type="text/javascript">
    function testAjax() {
        ajaxPost("/test_ajax", {'number': '5'},
            function (r) { alert(r.result); });
    }
</script>
…
<input type=button value="Test"
       onclick="testAjax();">
```

- `onclick` handler: calls `testAjax()` if the button is pressed.
- `ajaxPost()`: initiates Ajax request using URL `/test_ajax` and one argument `number` with the value `5`.
- the anonymous function processes Ajax request results (shows the popup message with `result` value of the web response).
- (there is also `ajaxGet()` function without input arguments).

More on Events and Responses

- There are many JavaScript events besides `onclick`: https://www.w3schools.com/jsref/dom_obj_event.asp
- For example, if we need to handle the changes in an input box:

```
<input type="text"
       onchange="testAjax(this.value);"/>
…
function testAjax(text) {...}
```


More on Events and Responses

- Alert windows are not common in modern web applications.
- A better approach is to display messages inside HTML by modifying special placeholder elements:

```
<script type="text/javascript">
  function testAjax() {
    ajaxPost("/test_ajax", {'number': '5'},
      function (r) {
        document.getElementById("id_message").
                          innerHTML = r.result;
      });
    }
</script>
…
<p id="id_message"></p>
```


Coming Exercise

It's time to wrap up everything we know,
and design the complete app with a rich user interface!