Обработка форм

GET / POST формы

GET - метод для **получения** данных. GET запросы могут быть закешированны промежуточными серверами. **GET** должен применяться только в **поисковых** формах.

POST - метод для изменения данных. POST запросы никогда не кешируются промежуточными серверами. **POST** должен применяться в формах, **изменяющих данные** на сервере.

Общий сценарий обработки

Best practice

- **Всегда** проверять пользовательские данные
- Для форм, изменяющих данные, использовать метод **POST**
- Не заставлять вводить данные повторно
- Сообщать об ошибках детально - по полям
- Сообщать об успешном сохранении формы
- При успешном сохранении делать перенаправление

HTTP Redirect

Перенаправления в HTTP

Перенаправления в HTTP

- `302 Found` - временное перенаправление
- `301 Moved Permanently` - постоянное перенаправление (кешируется в браузере)
- `Location: url` - URL для повторного запроса. Может быть как абсолютным, так и относительным.

Перенаправления в Django

```
from django.http import HttpResponseRedirect

def some_view(request):
    # logic..
    return HttpResponseRedirect('/new_url/')

# уязвимость open redirect
def dangerous_view(request):
    url = request.GET.get('continue')
    return HttpResponseRedirect(url)
```

Django forms

Описание форм

```
from django import forms

class FeedbackForm(forms.Form):
    email = forms.EmailField(max_length=100)
    message = forms.CharField(widget=forms.Textarea)

    def clean(self):
        if is_spam(self.cleaned_data):
            raise forms.ValidationError(
                u'Сообщение похоже на спам',
                code='spam'
            )
```

```
class AddPostForm(forms.Form):
    title = forms.CharField(max_length=100)
    message = forms.CharField(widget=forms.Textarea)

    def clean_message(self):
        message = self.cleaned_data['message']
        if not is_ethic(message):
            raise forms.ValidationError(
                u'Сообщение не корректно', code=12)
        return message + \
               "\nThank you for your attention."

    def save(self):
        post = Post(**self.cleaned_data)
        post.save()
        return post
```

Типы полей

- `BooleanField` - флажок
- `CharField` - текстовое поле ввода
- `EmailField` - текстовое поле, Email
- `ChoiceField` - выбор из нескольких вариантов
- `DateField` - выбор даты
- `DateTimeField` - выбор даты и времени
- `FileField` - загрузка файлов

Валидация данных

- По типу поля, например `EmailField`
- `clean_xxx` - доп. проверка поля xxx, может изменить значение
- `clean` - доп. проверка всех полей формы

Методы `clean` и `clean_xxx` должны использовать `self.cleaned_data` для получения данных формы и поднять `ValidationError` в случае некорректных данных.

Использование во view

```
def post_add(request):
    if request.method == "POST":
        form = AddPostForm(request.POST)
        if form.is_valid():
            post = form.save()
            url = post.get_url()
            return HttpResponseRedirect(url)
    else:
        form = AddPostForm()
    return render(request, 'blog/post_add.html', {
        'form': form
    })
```

Использование в шаблонах

```
{{ form.as_ul }}

{{ form.as_p }}

{{ form.as_table }}
```

```
{% for e in form.non_field_errors %}
  <div class="alert alert-danger">{{ e }}</div>
{% endfor %}
<form class="form-horizontal" method="post" action="/blog/add/">
  <fieldset>
    {% for field in form %}
      <div class="control-group
        {% if field.errors %}has-error{% endif %}">
        <label class="control-label">{{ field.label }}</label>
        <div class="controls">{{ field }}</div>
      </div>
    {% endfor %}
  </fieldset>
  <div class="form-actions">
    <button type="submit" class="btn btn-primary">
    Сохранить</button>
  </div>
</form>
```

Model forms

```
from django.forms import ModelForm

class ArticleForm(ModelForm):
    class Meta:
        model = Post
        fields = ['title', 'content',
                  'category', 'tags']
```

Метод `save` уже определен и сохраняет модель `Meta.model`

Безопасность

Проверка пользователя

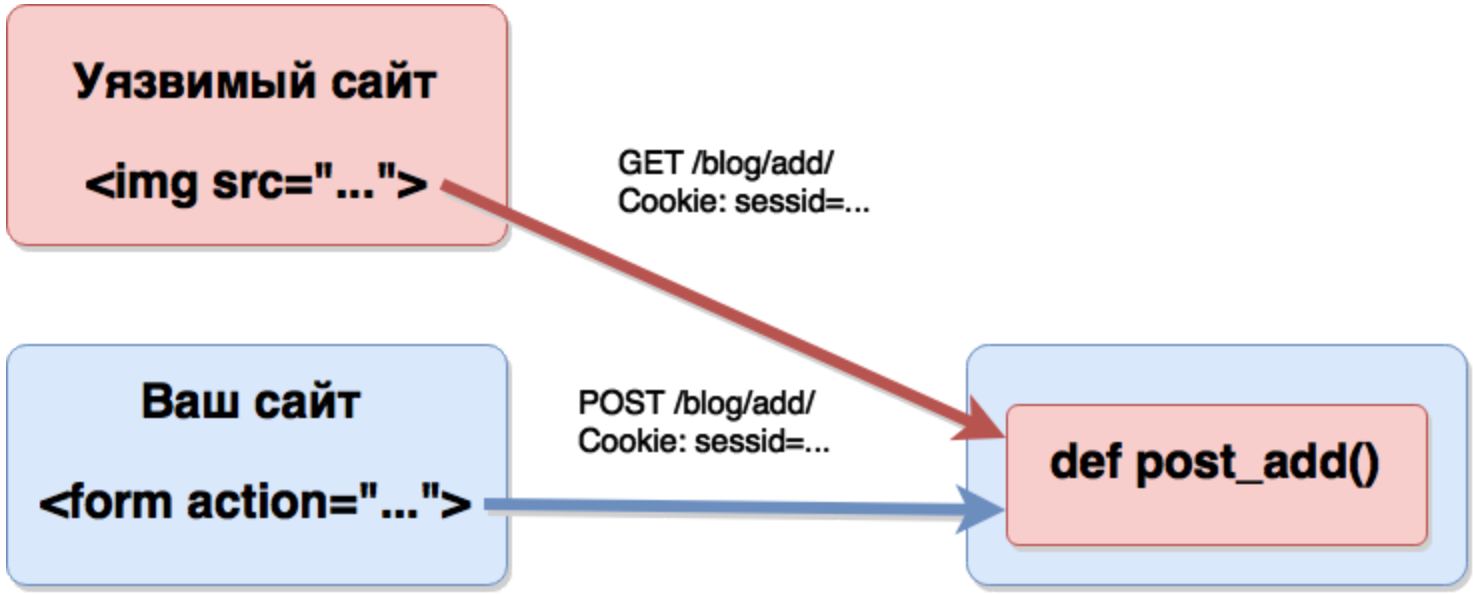
```
class AddPostForm(forms.Form):
    # ... поля ...
    def __init__(self, user, **kwargs):
        self._user = user
        super(AddPostForm, self).__init__(**kwargs)
    def clean(self):
        if self._user.is_banned:
            raise ValidationError(u'Доступ ограничен')
    def save(self):
        self.cleaned_data['author'] = self._user
        return Post.objects.create(**self.cleaned_data)
```

Проверка пользователя (2)

```
from django.contrib.auth.decorators \
    import login_required

@login_required
def post_add(request):
    form = AddPostForm(request.user, request.POST)
    if form.is_valid():
        post = form.save()
```

Cross Site Resource Forgery



Методы борьбы с CSRF

- Проверка метода `@require_POST`
- Проверка заголовка `Referer`
- Проверка CSRF-токенов

```
<form method="POST" action="/blog/add">
    {% csrf_token %}
</form>
```