

AJAX

# Asynchronous JavaScript And XML

**AJAX** - технология загрузки данных / отправки форм без обновления WEB страницы.

XML - совершенно не обязателен, возможны отправка и прием данных любого типа. Чаще всего вместо XML используется HTML либо JSON для загрузки сырых данных.

# AJAX на стороне клиента

```
var xhr = new XMLHttpRequest();
xhr.open('POST', '/xhr/test.html', true);
xhr.onreadystatechange = function() {
    if (xhr.readyState == 4) {
        if (xhr.status == 200) {
            alert(xhr.responseText);
        }
    }
};
xhr.send("a=5&b=4");
```

# AJAX при помощи jQuery

```
$.ajax({  
    url:    '/blog/comments/add/',  
    type:   'POST',  
    data:   { post_id: 12, text: 'Занятная идея!' },  
}).success(function(data) {  
    if (data.status == 'ok') {  
        console.log(data.comment_id);  
    }  
}).error(function() {  
    console.log('http error')  
});
```

# АJAX при помощи Fetch API

```
fetch('/blog/comments/add/', {  
  method: 'POST',  
  headers: {  
    'Content-Type': 'application/json',  
  },  
  body: JSON.stringify({ post_id: 12, text: 'Занятная идея!' }),  
})  
  .then((response) => response.json())  
  .then((data) => {  
    if (data.status == 'ok') {  
      console.log(data.comment_id);  
    }  
  });
```

# Особенности и ограничения AJAX

- **Same Origin Policy** - AJAX запросы можно отправлять только на свой домен. В современных браузерах есть CORS.
- Т.к. данные передаются явно в метод `send`, то **нельзя загружать файлы**. В HTML5 есть FormData.
- AJAX на стороне сервера не отличим от обычного запроса. jQuery добавляет заголовок `X-Requested-With: XMLHttpRequest`
- Результаты запроса передаются в JavaScript функцию поэтому стандартные методы обработки ошибок - не работают.

# Загрузка HTML данных

```
def comments_list(request):  
    post_id = request.GET.get("post_id")  
    post = get_object_or_404(Post, post_id)  
    comments = paginate(request, post.comments)  
    return render(request, "blog/comments.html", {  
        "comments": comments  
    })
```

В `blog/comments.html` отображается только HTML код комментариев, без окружающей страницы.

# Обмен данными в JSON

```
{  
    "status": "ok",  
    "comment_id": 123  
}
```

```
{  
    "status": "error",  
    "code": "no_auth",  
    "message": "вы не авторизованы"  
}
```



# HttpResponseAjax

```
from django.http import JsonResponse
```

```
class HttpResponseAjax(JsonResponse):  
    def __init__(self, status="ok", **kwargs):  
        kwargs["status"] = status  
        super().__init__(kwargs)
```

```
class HttpResponseAjaxError(HttpResponseAjax):  
    def __init__(self, code, message):  
        super().__init__(  
            status="error", code=code, message=message  
        )
```

# Использование HttpResponseRedirect

```
@login_required_ajax
def comment_add(request):
    form = AddCommentForm(request.POST)
    if form.is_valid():
        comment = form.save()
        return HttpResponseRedirect(comment_id=comment.id)
    else:
        return HttpResponseRedirectError(
            code="bad_params",
            message=form.errors.as_text(),
        )
```

# Проверка авторизации в AJAX

```
def login_required_ajax(view):  
    def decorated_view(request, *args, **kwargs):  
        if request.user.is_authenticated():  
            return view(request, *args, **kwargs)  
        else:  
            return HttpResponseAjaxError(  
                code="no_auth",  
                message="Требуется авторизация",  
            )  
    return decorated_view
```

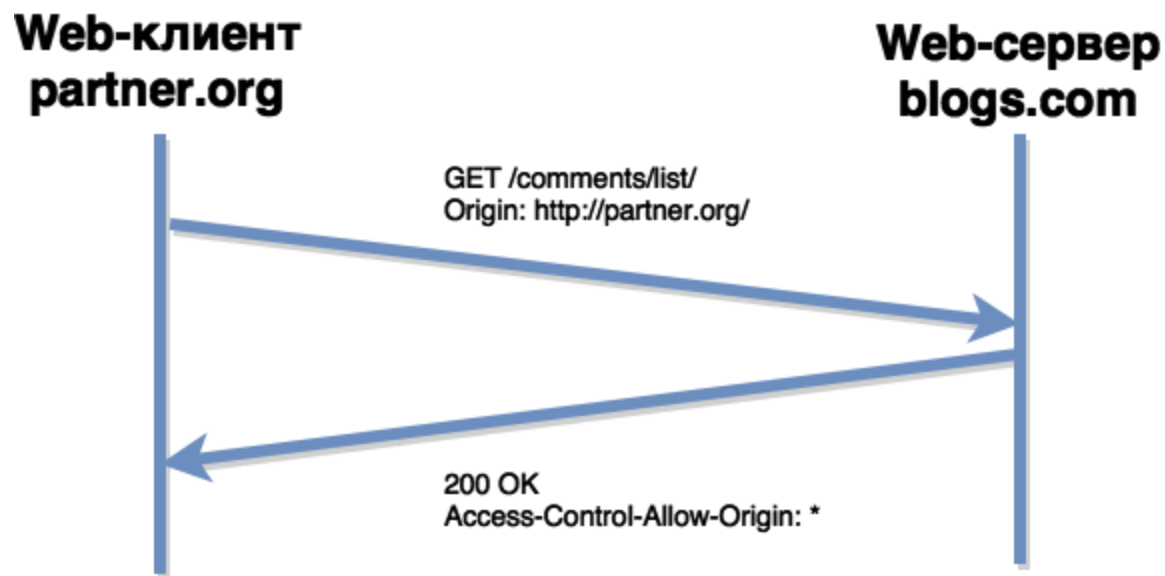
CORS

# Cross Origin Resource Sharing

Браузер выполняет AJAX запросы даже к чужому домену, но в этом случае не вызывает функцию-callback в JavaScript, т.е. не дает использовать данные, загруженные с чужого домена.

**CORS** позволяет серверу явно разрешить использование данных при кросс-доменных запросах.

# Cross Origin Resource Sharing



# Заголовки CORS

- `Origin` - указывает URL запрашивающего приложения
- `Access-Control-Allow-Origin: origin` - разрешает использовать данные в кросс-доменном запросе. `origin` должен либо совпадать с заголовком `Origin` в запросе, либо `*`.
- `Access-Control-Allow-Credentials` - позволяет использовать данные, если были переданы cookies.

# Использование CORS

Не следует разрешать CORS для всех запросов ( \* ).

Хорошей практикой является:

- Проверка суффикса домена
- Проверка домена по списку доверенных



# CORS на клиенте

```
$.ajax({  
  url: 'https://site.com/blog/comments/add/',  
  type: 'POST',  
  data: { post_id: 12, text: 'Занятная идея!' },  
  crossDomain: true,  
  xhrFields: {  
    withCredentials: true  
  }  
}).success(function(data) {  
  // ...  
});
```

# CORS на сервере

```
def allow_cors(view):  
    def decorated_view(request, *args, **kwargs):  
        response = view(request, *args, **kwargs)  
        origin = request.META.get("HTTP_ORIGIN")  
        if not origin:  
            return response  
        for domain in settings.CORS_WHITE_LIST:  
            if origin.endswith("." + domain):  
                response["Access-Control-Allow-Origin"] = origin  
                response["Access-Control-Allow-Credentials"] = "true"  
        return response  
    return decorated_view
```