Web Technologies

Lecture 11

Implementing RESTful services

REST Services

- **RE**presentational **S**tate **T**ransfer
- **Stateless**
- **Stardard design architecture** for web APIs
  - Allows to publicly expose methods over the Internet to be accessed and manipulated outside the program itself
    - Web services
  - Get data from an application without having to visit the application itself (e.g., go to a particular website)
  - Achieved through RESTful URIs
    - Example: *GET /mycollection/{id}*

Building a RESTful service

- Use **a single URL for the requests**
  - Avoids having multiple URLs
    - increases maintability
  - Achived through *.htaccess* rewrite rules
- **Handle cross domain requests** to the unique URL
  - Receive HTTP requests
  - Extract endpoint from URI
  - Detect HTTP method (GET, POST, PUT, DELETE)
  - Assemble additional data provided in the header or URI
  - Pass information to proper method for processing
  - Send back HTTP response

Cross Origin Resource Sharing

- CORS W3C specification
- Enables cross domain communication
  - Javascript for instance explicitly prohibits this
    - See AJAX
- Client and server side
  - Requires coordination between the two
- Built on top of *XMLHttpRequest*
- Server adds some headers allowing client to access its data

NOTE

- Not a substitute for sound security practices!
- It only allows cross domain access

CORS browser support

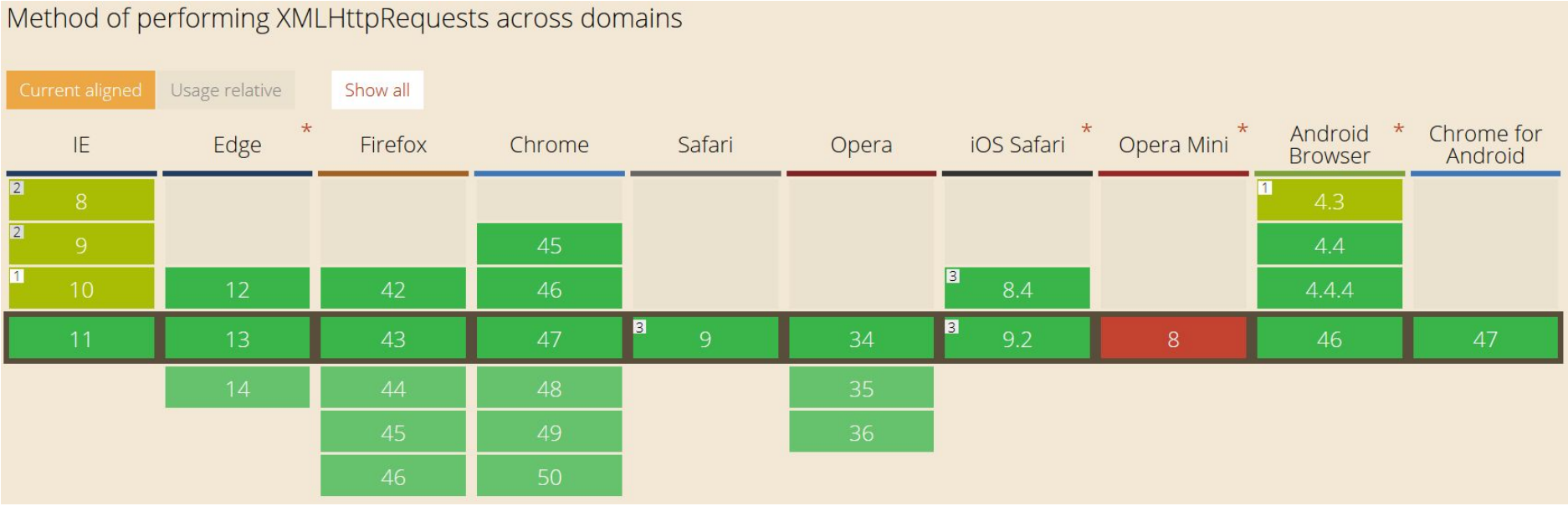


Image source: *http://caniuse.com/#search=cors*

CORS on the client side

1. Create CORS request

```
// Create the XHR object.
function createCORSRequest(method, url) {
  var xhr = new XMLHttpRequest();
  if ("withCredentials" in xhr) {
    // XHR for Chrome/Firefox/Opera/Safari.
    xhr.open(method, url, true);
  } else if (typeof XDomainRequest != "undefined") {
    // XDomainRequest for IE.
    xhr = new XDomainRequest();
    xhr.open(method, url);
  } else {
    // CORS not supported.
    xhr = null;
  }
  return xhr;
}
```

2. Send the CORS request

```
// Make the actual CORS request.
function makeCorsRequest() {
  // All HTML5 Rocks properties support CORS.
  var url = 'http://updates.html5rocks.com';

  var xhr = createCORSRequest('GET', url);
  if (!xhr) {
    alert('CORS not supported');
    return;
  }

  // Response handlers.
  xhr.onload = function() {
    var text = xhr.responseText;
    var title = getTitle(text);
    alert('Response from CORS request to ' + url + ': ' + title);
  };

  xhr.onerror = function() {
    alert('Woops, there was an error making the request.');
  };

  xhr.send();
}
```

A valid CORS request always contains an *Origin* header

POST /receiver HTTP/1.1
Origin: http://www.issuer.com

The header is added by the browser, and cannot be controlled by the user

CORS on the server side

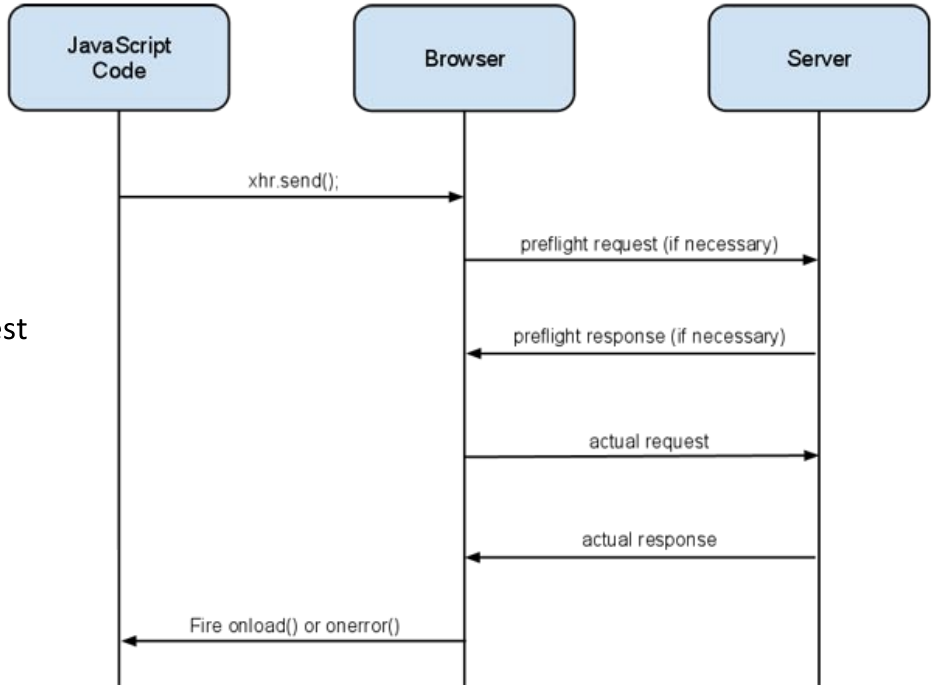
- Requires to set up some response headers
  - **Access-Control-Allow-Origin**: mandatory
  - **Access-Control-Allow-Methods**: list of supported methods. Mandatory
  - **Access-Control-Allow-Credentials**: for cookies. Works in conjunction with the withCredentials property on the XMLHttpRequest 2 object
  - **Access-Control-Expose-Headers**: allows access to various response headers
  - **Access-Control-Request-Method**: the request method. Always present
  - **Access-Control-Request-Headers**: list of request headers
  - **Access-Control-Max-Age**: allows preflight response to be cahced

```
header("Access-Control-Allow-Origin: *");
header("Access-Control-Allow-Methods: *");
header("Content-Type: application/json");
```

PUT and DELET require a **preflight request**

- Extracommunication with the server
- Asks permission to make the actual request
- Can be cached to avoid making it for every request

Source: *http://www.html5rocks.com/en/tutorials/cors/*



The .htaccess file

```
<IfModule mod_rewrite.c>
    RewriteEngine On
    RewriteCond %{REQUEST_FILENAME} !-f
    RewriteCond %{REQUEST_FILENAME} !-d
    RewriteRule api/v1/(.*)$ api/v1/api.php?request=$1 [QSA,NC,L]
</IfModule>
```

1. *Check the existence of mod_rewrite*
2. *Activate rewrite engine*
3. *For any request that does not point to an existing file or directory forward it to api.php by using the paramter request*

QSA: named capture is appended at the end of the URI

NC: URIs are not case sensitive

L: stop after processing this rule

API class

- PHP implementation
- JSON communication format
- Abstract class defining the basic wrapper for the various endpoints the API will be using
- Concrete class implementing endpoint functionality
- Enable secure & cross domain access

API abstract class

```
abstract class API {
        protected $method = ''; //PUT, GET, POST, DELETE
        protected $endpoint = ''; //e.g.: /files
        protected $verb = ''; //e.g.: /files/process
        protected $args = Array();  //e.g.: /files/process/23
        protected $file = Null; // input of the PUT request. Can be JSON text
        public function __construct($request) {
                header("Access-Control-Allow-Orgin: *");
                header("Access-Control-Allow-Methods: *");
                header("Content-Type: application/json");
                //extract method, endpoint, verb, args
                $this->args = explode('/', rtrim($request, '/'));
                $this->endpoint = array_shift($this->args);
                if (array_key_exists(0, $this->args) && !is_numeric($this->args[0])) {
                        $this->verb = array_shift($this->args);
                }
                $this->method = $_SERVER['REQUEST_METHOD'];
                // search for PUT and DELETE in HTTP_X_HTTP_METHOD
                if ($this->method == 'POST' && array_key_exists('HTTP_X_HTTP_METHOD', $_SERVER)) {
                        if ($_SERVER['HTTP_X_HTTP_METHOD'] == 'DELETE') {
                                $this->method = 'DELETE';
                        } else if ($_SERVER['HTTP_X_HTTP_METHOD'] == 'PUT') {
                                $this->method = 'PUT';
                        } else {
                                throw new Exception("Unexpected Header");
                        }
                }
                switch ($this->method) {
                        case 'DELETE':
                        case 'POST':  $this->request = $this->_cleanInputs($_POST); break;
                        case 'GET':   $this->request = $this->_cleanInputs($_GET); break;
                        case 'PUT':   $this->request = $this->_cleanInputs($_GET);
                                      $this->file = file_get_contents("php://input"); break;  // read raw post data
                        default:      $this->_response('Invalid Method', 405); break;
                }
        }
        […] // see next slide
}
```

API abstract class

- Private methods
  - **_response**
    - Sets the response header and body
  - **_cleanInputs**
    - Stores the request variables in a key-value list
  - **_requestStatus**
    - Returns a message associated with a particular code
    - Used by _response
- Public method
  - **processApi**
    - Determine if the concrete class implements the endpoint method called by the client

```
public function processAPI() {
        if (method_exists($this, $this->endpoint)) {
                // call the endpoint method
                return $this->_response($this->{$this->endpoint}($this->args));
        }
        return $this->_response("No Endpoint: $this->endpoint", 404);
}
```

API concrete class

- CORS opens up a huge security vulnerability
  - Must ensure that only certain clients with a unique key can access the API

```
require_once 'API.class.php';
class MyAPI extends API {
        protected $token;
        public function __construct($request, $origin) {
                parent::__construct($request);
                $User = new User()
                if (!array_key_exists('apiKey', $this->request)) {
                        throw new Exception('No API Key provided');
                } else if (!$APIKey->verifyKey($this->request['apiKey'], $origin)) {
                        throw new Exception('Invalid API Key');
                } else if (!array_key_exists('token', $this->request)) {
                        throw new Exception('No User Token provided');
                }
                $this->token = $this->request['token']);
        }
        // Example of an Endpoint. Called by processAPI (see previous slide)
        protected function returnToken() {
                if ($this->method == 'GET') {
                        return "Your token is " . $this->token;
                } else {
                        return "Only accepts GET requests";
                 }
        }
}
```

Using the API

```
// Requests from the same server don't have a HTTP_ORIGIN header
if (!array_key_exists('HTTP_ORIGIN', $_SERVER)) {
	$_SERVER['HTTP_ORIGIN'] = $_SERVER['SERVER_NAME'];
}
try {
	$API = new MyAPI($_REQUEST['request'], $_SERVER['HTTP_ORIGIN']);
	echo $API->processAPI();
} catch (Exception $e) {
	echo json_encode(Array('error' => $e->getMessage()));
}
```

Note: http://coreymaynard.com/blog/creating-a-restful-api-with-php/

What’s next?

- Node.js
- Cloud computing