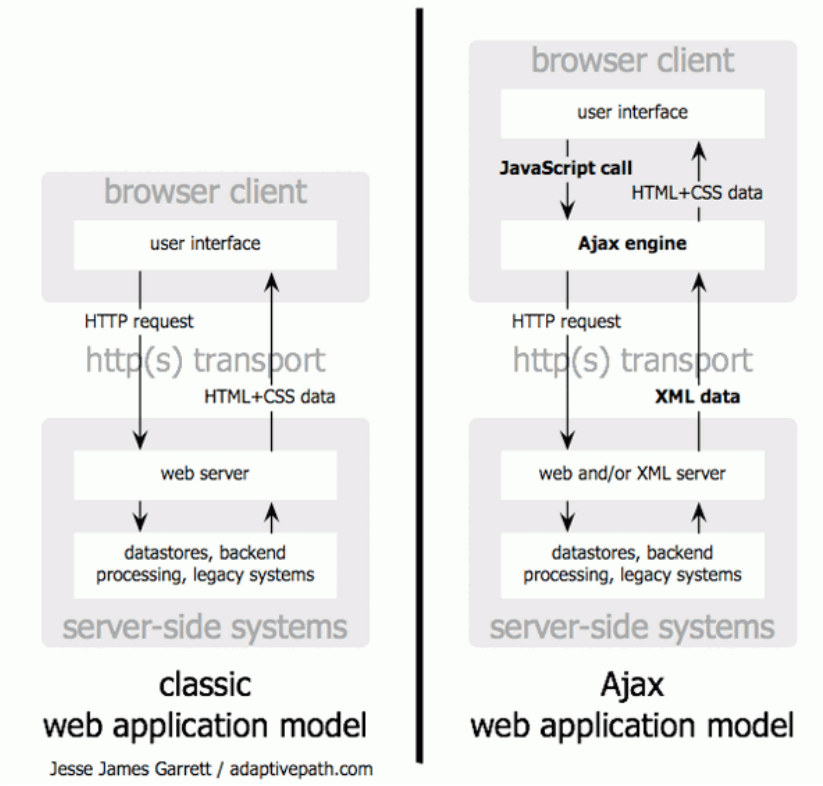
Laborator 10 - AJAX

Obiective: În urma parcurgerii acestui laborator studentul va:

- înţelege conceptele şi protocoalele din spatele AJAX
- putea implementa cereri AJAX atât folosind "vanilla" Javascript, cât şi jQuery

AJAX

Termenul **AJAX** descrie o arhitectură pentru aplicaţii web bazată pe HTTP scripting şi obiectul XMLHttpRequest. Termenul a fost creat de Jesse James Garret în eseul său din februarie 2005. Punctul cheie al acestei abordări este faptul că paginile web nu trebuie reîncărcate complet iar datele interschimbate au de obicei un volum mic. Efectul este o comportare a aplicaţiilor web mult îmbunătăţită din puctul de vedere al experienţei utilizatorului.



AJAX poate fi formalizat într-un mecanism RPC (spre exemplu), însă în cele mai multe cazuri se preferă soluţii mai simple şi mai rapide. De multe ori, XML nici măcar nu mai este folosit, preferându-se formate mai lightweight (JSON sau chiar plain-text).

Implementare AJAX

XMLHTTPRequest

Obiectul XMLHTTPRequest a fost introdus pentru a se putea controla protocolul HTTP din Javascript sincron şi asincron, fără a se apela la artificii de genul `<iframe>`-urilor. Din păcate, obiectul nu a fost niciodată standardizat şi de aceea întâlnim diferite implementări în diferite browsere (IE, of course).

```
// normal browser (IE 7 included)
var request = new XMLHttpRequest();
// IE 5 or 6
var request = new ActiveXObject("Msxml2.XMLHTTP");
// or, depending on installed client libraries
var request = new ActiveXObject("Microsoft.XMLHTTP");
```

De obicei, se crează metode care abstractizează aceste diferenţe de instanţiere. Un exemplu este prezentat mai jos:

```
var HTTP = {
  // This is a list of XMLHttpRequest-creation factory functions to try
  _factories: [
```


Search

- Repartizare teme
- Catalog PW

Laboratoare

- Laborator 01 - Introducere
- Laborator 02 - BD in PHP
- Laborator 03 - Arrays, Magic Methods
- Laborator 04 - (X)HTML, CSS
- Laborator 05 - Formulare HTML, Persistența Datelor
- Laborator 06 - Securitate I
- Laborator 07 - Securitate II
- Laborator 08 - JavaScript
- Laborator 09 - DOM scripting
- Laborator 10 - AJAX
- Laborator 11 - Web Frameworks

Teme

- Tema 01 - API pentru BD
- Tema 02 - Template System & Controller
- Tema 03 - Generator de formulare
- Tema 04 - Tree Web UI

Login

```
  function() { return new XMLHttpRequest(); },
  function() { return new ActiveXObject("Msxml2.XMLHTTP"); },
  function() { return new ActiveXObject("Microsoft.XMLHTTP"); }
  ],
  // When we find a factory that works, store it here.
  _factory: null,
  // Create and return a new XMLHttpRequest object.
        //
        // The first time we're called, try the list of factory functions until
        // we find one that returns a non-null value and does not throw an
        // exception. Once we find a working factory, remember it for later use.
        //
  newRequest: function() {
    if (HTTP._factory != null) return HTTP._factory();

    for(var i = 0; i < HTTP._factories.length; i++) {
     try {
      var factory = HTTP._factories[i];
      var request = factory();
      if (request != null) {
       HTTP._factory = factory;
       return request;
      }
     }
     catch(e) {
      continue;
     }
    }
                        // If we get here, none of the factory candidates succeeded,
                        // so throw an exception now and for all future calls.
    HTTP._factory = function() {
     throw new Error("XMLHttpRequest not supported");
     }
    HTTP._factory ();
   }
};
```

Observatii:

- exemplul de mai sus aplica o varianta a *design pattern-ului* ***Singleton*** pentru a obtine o cerere XMLHttp; (foarte asemanator cu SingletonDB, de la tema1);
- observati membrul **_factory** al clasei HTTP: acesta este initial `null`, insa va retine un obiect de tip **XMLHttpRequest**;
- observati functia **newRequest**:
  - daca **_factory** a fost creat, functia il intoarce;
  - altfel, se incearca succesiv apelurile retinute in vectorul **_factories**; in functie de browser, unul din aceste apeluri va avea succes (altfel, se va arunca exceptie); rezultatul apelului care a avut succes este retinut in **_factory**, si intors de functie;

Trimiterea unei cereri cu XMLHttpRequest

După ce crearea unui astfel de obiect, următoarele metode sunt puse la dispoziţie pentru construirea unei cereri catre server:

- `request.open(`*method*`, `*url*`, `*asynchronous*`, [`*username*`, `*password*`])`
- `request.setRequestHeader(`*name*`, `*value*`)`
- `request.send(`*content*`)`

Răspunsul este stocat în cadrul **aceluiaşi obiect**. Inspectând următoarele metode şi proprietăţi se poate verifica şi manipula răspunsul:

- **request.status**: intoarce codul HTTP asociat raspunsului (poate fi `200` pentru *succes*, `404` daca pagina nu exista, etc)
- **request.statusText**: intoarce un text asociat codului HTTP din raspuns;
- **request.responseText**: intoarce raspunsul efectiv al cererii HTTP;
- **request.getResponseHeader()**: intoarce **header-ul** raspunsului;

Cereri sincrone

O cerere **sincronă** se efectuează atunci când cel de-al treilea parametru al metodei **open** este **false**. Acest lucru **determină blocarea execuţiei de cod Javascript până în momentul primirii răspunsului de la server**. Secvenţa de cod de mai jos exemplifică o cerere sincronă:

```
var request = HTTP.newRequest();

request.open("GET", url, false); //prepare a HTTP request using GET method, to 'url'; request
request.setRequestHeader("User-Agent", "XMLHttpRequest"); //set header fields
request.setRequestHeader("Accept-Language", "en");
request.setRequestHeader("If-Modified-Since", lastRequestTime.toString());
request.send(null);//send an empty request; the request is blocking;
```

```
//after send method returns, check response
if (request.status == 200) {
    // We got the server's response. Display the response text.
    alert(request.responseText);
}
else {
    // Something went wrong. Display error code and error message.
    alert("Error " + request.status + ": " + request.statusText);
}

// or, for XML responses
if (request.status == 200) {  // Make sure there were no errors
    // Make sure the response is an XML document
    if (request.getResponseHeader("Content-Type") == "text/xml") {
        var doc = request.responseXML;
        // Now do something with the response document
    }
}
```

Observatii:

- din cauza ca cererea este **sincrona**, apelul **send** din exemplul de mai sus este **blocant**; functia va intoarce doar dupa primirea raspunsului de la server;
- apoi, se poate inspecta codul HTTP (folosind membrul **status**), continutul si/sau header-ul raspunsului;

Cereri asincrone

In momentul in care raspunsul unei cereri HTTP realizata folosind XMLHttpRequest intarzie, sau ajunge mai greu (din diverse motive), pagina se blocheaza (din cauza apelului blocant al send). Acest lucru poate fi neplacut. Tocmai de aceea, din punct de vedere al experienței utilizatorului, este mult mai bine ca aceste cereri să fie **non-blocante (asincrone)**. In aceasta situatie, se pune problema **depistarii momentului in care cererea este primita** (si executarea unei actiuni corespunzatoare, pe baza raspunsului).

Pentru aceasta avem la dispozitie urmatoarele:

- **readystate**: este o proprietate a obiectului XMLHttpRequest, si contine **starea** cererii curente:
  - 0: `open()` încă nu a fost apelat
  - 1: `open()` a fost apelat, dar `send()` încă nu a fost apelat
  - 2: `send()` a fost apelat, dar serverul nu a răspuns încă
  - 3: se primesc date de la server (atenție, implementările diferă între browsere aici)
  - 4: toate datele au fost primite de la server
- **onreadystatechange**: este un eveniment; i se atribuie o functie, care este apelata de cate ori **starea** cererii se modifica;

Secvența de cod de mai jos prezintă o cerere asincronă (AJAX):

```
// Create an XMLHttpRequest using the utility defined earlier
var request = HTTP.newRequest();

// Register an event handler to receive asynchronous notifications.
// This code says what to do with the response, and it appears in a nested
// function here before we have even submitted the request.
request.onreadystatechange = function() {
    if (request.readyState == 4) {  // If the request is finished
        if (request.status == 200)  // If it was successful
            alert(request.responseText);  // Display the server's response
    }
}

// Make a GET request for a given URL. We don't pass a third argument,
// so this is an asynchronous request
request.open("GET", url);

// We could set additional request headers here if we needed to.

// Now send the request. Since it is a GET request, we pass null for
// the body. Since it is asynchronous, send() does not block but
// returns immediately.
request.send(null);
```

Same-origin policy

Această politică de securitate restricţionează conţinutul cu care poate interacţiona codul Javascript la orginea documentului din contextul căruia se execută scriptul. Originea documentului este definită ca o combinaţie între **protocol, host şi port** (determinate prin URL). Este important să reţineţi că orginea

este legată de document şi nu de script: puteţi avea un script încărcat de pe alt domeniu din care să efectueze cereri către originea documentului vostru. Se pot face excepţii pentru cererile între subdomenii dacă se foloseşte proprietatea `domain` a obiectului `document`. Aici găsiţi o explicaţie mai detaliată.

Această implementare de securitate este necesară pentru a preveni, printre altele, furtul de informaţie. Gândiţi-vă la posibilitatea ca un site care rulează cod maliţios să deschidă şi să aibă acces la un alt tab în care aveţi afişate informaţii sensibile.

În cazul particular al folosirii obiectului XMLHttpRequest, s-ar referi la faptul că se pot face cereri doar peste HTTP şi doar către locul din care originat documentul din care a pornit cererea. Totuşi, cererile AJAX cross-domain pot avea o utilitate foarte mare - nu ar mai fi nevoie să se facă proxying server-side pentru a putea servi conţinut de pe alte domenii. Soluţia la care s-a ajuns este folosirea headerelor `Origin: [domeniu emitent]` de către cel ce emite cererea şi `Access-Control-Allow-Origin: [* / lista acceptată de domenii, separate prin ,]`. Dacă headerul `Access-Control-Allow-Origin` conţine domeniul emitent (sau *) atunci browserul primeşte şi interpretează răspunsul. Mai multe detalii puteţi vedea aici.

AJAX în jQuery

jQuery oferă metode care facilitează efectuarea acţiunilor AJAX. Metodele sunt bine documentate şi demonstrate pe site-ul cu documentaţia.

Este mult mai facilă folosirea jQuery decât crearea de obiecte care să abstractizeze `XMLHttpRequest` etc. Trebuie însă avut în vedere faptul că fără o înţelegere bună a protocolului HTTP, a limbajului şi a modului de funcţionare a browserelor este mult mai greu să scrieţi cod de calitate. Exemplul de mai jos demonstrează folosirea jQuery pentru a face cereri peste HTTP:

```
// on page load
$(document).ready(function () {
 // reference a button
 var button = $(...);
       // add an event handler
 button.click(function () {
  // do an ajax request
  $.ajax({
   // specify the url to request
   url: 'document.html',
   // request method
   type: 'GET',
   // dataType, can be used for automated parsing
   dataType: 'html',
   // timeout for this request
   timeout: 1000,
   // error callback
   error: function(){
    alert('Error loading HTML document');
   },
   // successful request callback
   success: function(html){
    // do something with html
   }
  });
 });
});
```

Taskuri

Descărcaţi arhiva de aici şi dezarhivaţi-o într-un folder de pe server.

1. În pagina task1.html, adăugaţi tagurilor `span` care au setată clasa `tooltip` următoarea funcţionalitate:
   - la mouseover, se efectuează o cerere AJAX (post) la `script/wordJSON.php`
   - se trimite un JSON de forma: `'{ "word" : "test" }'`
   - se primeşte definiţia cuvântului, tot JSON: `'{ "definition" : "definition for test" }'`
   - definiţia va fi afişată undeva în pagină (orice variantă e acceptabilă, mai puţin cele care folosesc `document.write()`) **(4p)**
2. În pagina task2.html, adăugaţi tagurilor `span` care au setată clasa `tooltip` următoarea funcţionalitate:
   - la mouseover, se efectuează o cerere AJAX (post) la `script/wordXML.php`
   - se trimite un XML de forma:

     ```
     <content>
       <word>test</word>
     </content>
     ```

   - se primeşte un XML de forma:

     ```
     <content>
       <definition>definition for test</definition>
     </content>
     ```

- definiţia este afişată în dreptul cursorului (la fel cum se întâmplă la tooltipurile default din browser). **(4p)**

3. La fel ca la punctul 1, însă folosind jQuery. **(2p)**
4. La fel ca la punctul 2, însă folosing jQuery. **(2p)**

Bonus:

5. Implementaţi unul sau ambele taskuri de jQuery ca pluginuri. **(2p)**

Show pagesource Old revisions Back to top