PAU CASESNOVES
Centre Integrat de Formació Professional

Desenvolupament d'aplicacions web

Desenvolupament web a l'entorn client

UT 7.2: HTTP i Rest.

Joan Pons Tugores

Índex de continguts

Rest Web Services i HTTP

Quan volem comunicar dues màquines entre si per que intercanviïn informació necessitam establir unes regles, un protocol, que determinin com s'han de comunicar.

Quan feim una aplicació web les dues màquines són el navegador, el client, i el servidor. La web utilitza el protocol HTTP per comunicar-los. El client fa una petició al servidor en aquest protocol, per carregar una pàgina per exemple, i el servidor li envia la resposta utilitzant aquest mateix protocol.

Les aplicacions que utilitzen AJAX fan peticions als servidor per demanar dades i mostrar-les al client sense haver de recarregar tota la pàgina. O envien dades al servidor per modificar els seus recursos: eliminar dades de la base de dades, modificar-les, ...

Aquestes comunicacions s'han de fer seguint un protocol. Es podria haver creat un protocol nou, però es va decidir utilitzar un protocol ja existent que no implicaria fer canvis en els clients ni en el servidor: L'HTTP.

REST (REpresentational State Transfer) és una manera senzilla d'organitzar interaccions entre sistemes independents. Permet interactuar amb clients diversos com mòbils o altres llocs web. Es basa en el protocol HTTP.

Cada acció que el client vulgui demanar al servidor s'indicarà amb una URL. El servidor sabrà que en rebre una determinada URL no haurà de servir una pàgina html o una imatge sinó que haurà de realitzar certa acció i tornar una resposta al client.

Seqüència d'una petició

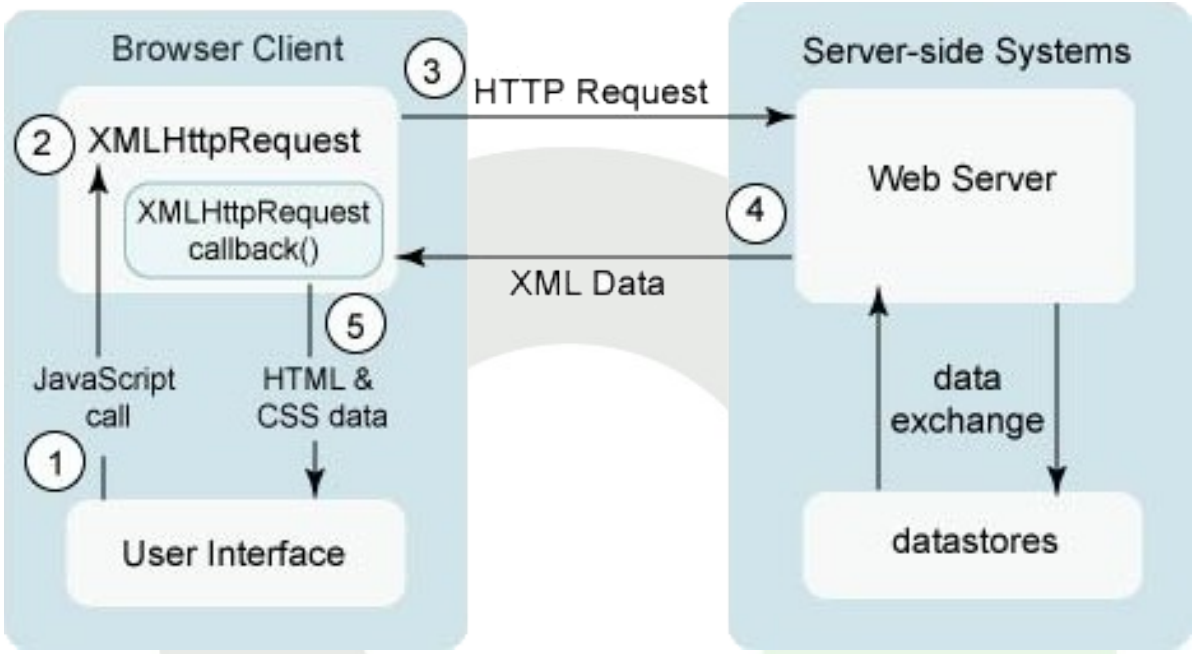

Figura 1: Seqüència d'una petició a un servei web

1. L'usuari provoca una crida al servidor.
2. Es crea l'objecte Javascript que farà la petició.
3. Es realitza la petició al servidor.
4. El servidor la processa i torna la resposta.
5. En rebre la resposta, el client executa la funció assignada a l'objecte. La funció interpretarà les dades i normalment provocarà la modificació de l'HTML i el CSS que veu l'usuari.

Conceptes necessaris

URL (Uniform Resource Locator)

Una URL identifica un recurs a la web de forma única, per exemple una pàgina web o fins i tot una part d'una pàgina web.

Una exemple d'URL podria ser

```
http://www.paucasesnoves.cat:8080/alumnes/notes.html?id=33&lang=ca#resum
```

Una URL té les següents parts:

- **Esquema** o **protocol**: Indica com es comunicaran el client i el servidor. Pot ser http, https, mailto, ftp, ... Es separa de la resta de la url per *://*

  ```
  http
  ```

- **Domini**: L'identificador del servidor. Normalment és un nom de domini, però també podria ser una IP

  ```
  www.paucasesnoves.cat
  ```

- **Port**: Separat del domini per : Si s'utilitzen els ports per defecte no es necessari posar-lo.

  ```
  8080
  ```

- **Authority**: El domini més el port

  ```
  www.paucasesnoves.cat:8080
  ```

- **Path**: Ruta al recurs dins del servidor

```
/alumnes/nota.html
```

- **Query string** o **search**: informació passada a la pàgina. Inclou ?

```
? id=33&lang=ca
```

- **Paràmetres**: Les dades que es passen al query string. Cada paràmetre té un nom i un valor. Els paràmetres es separen amb & A l'exemple anterior hi ha els paràmetres id i *lang*.

```
Id=33
lang=ca
```

- **Àncora**: Un identificador d'un element del document. El navegador mostrarà aquest document en pantalla, encara que hagi de fer scroll.

```
#resum
```

A les aplicacions REST una URL pot identificar per exemple els clients o un client determinat:

```
http://www.canostra.com/clients
http://www.canostra.com/clients/Toni
```

No es sol utilitzar la query string.

La URL ha de ser tan precisa com faci falta per identificar el recurs que volem utilitzar.

HTTP

HTTP és el protocol utilitzat per demanar i servir documents a la web. Considera dos rols ben diferenciats: el servidor i el client. Normalment és el client el que comença la interacció i el servidor respon.

És un protocol basat en text. Els missatges HTTP tenen una capçalera i un cos.

- El cos conté les dades que es volen transmetre per ser utilitzades segons el que digui la capçalera. Pot estar buid.
- La capçalera conté metadades com la codificació del text o el més important, l'operació HTTP. **Moltes vegades en fer serveis web REST és més important la capçalera que el cos del missatge**.

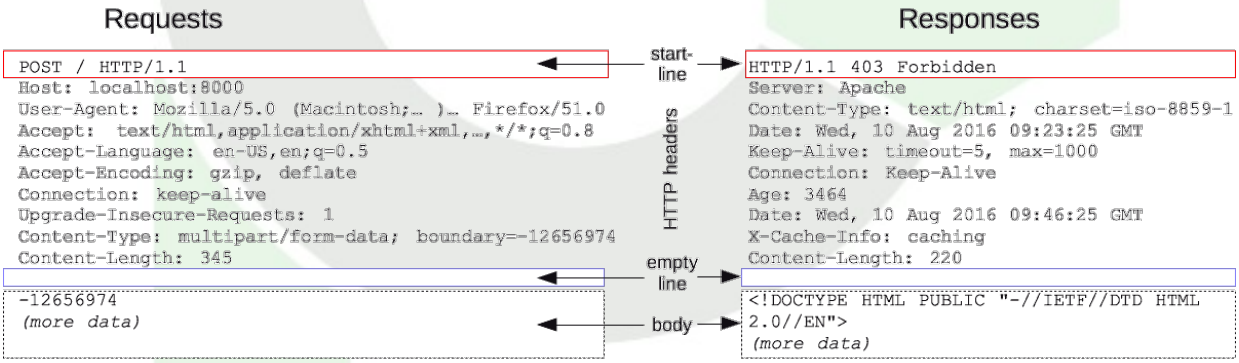

Figura 2: https://developer.mozilla.org/en-US/docs/Web/HTTP/Messages

Verbs o mètodes

Cada petició *HTTP* inclou a la capçalera un mètode, o verb, que indica al servidor que ha de fer amb el recurs indicat a la URL. Els més habituals són:

- **GET** Demana al servidor que envii les dades del recurs al client. En principi les dades no s'haurien de modificar al servidor com a resposta a un GET. És el mètode utilitzat per els navegadors quan demanen una web al servidor.

```
//Torna l'autor amb identificador 1
GET http://localhost/autors/1
```

  En Javascript:

```
req.open("GET", url + "autors/" + idAutor);
//Demana el resultat en json
req.setRequestHeader("Accept", "application/json");
req.send();
```

- **PUT** crea o modifica el recurs identificat a l'URL al servidor. El cos de la petició conté les dades a utilitzar. Res indica al servidor com ho ha de fer. Normalment s'utilitza per fer *updates* ja que necessita l'URL completa del recurs.

```
//Modifica l'autor amb identificador 1.
//Les dades de l'autor van dins el cos de la petició, per exemple un json
PUT http://localhost/autors/1
```

  En Javascript el que passem al mètode *send* s'inclou al cos de la petició:

```
req.open("PUT", url + "autors/111");
//Informa al servidor que les dades arriben en format JSON
req.setRequestHeader("Content-Type", "application/json");
req.send('{"idAut":111,"nomAut":"DIEZ RODRIGUEZ,MIQUELA",
```

```
            "dnaixAut":null,"imgAut":null, "fkNacionalitat":null}');
```

- **DELETE** esborra del servidor el recurs identificat per l'URL.

```
 //Esborra l'autor amb identificador 1
DELETE http://localhost/autors/1
```

En Javascript:

```
req.open("DELETE", url + "autors/" + idAutor);
req.send();
```

- **POST** processa les dades del cos de la petició com un element subordinat de la URL de la petició. Per exemple POST /clients/ crearà o modificarà dins /clients/ un recurs amb les dades de la petició. Cada vegada que s'executi farà un recurs nou. Normalment s'utilitza per els inserts.

```
//Afegeix un autor nou
//Les dades de l'autor van dins el cos de la petició, per exemple un json
POST http://localhost/autors
```

En Javascript el que passem al mètode *send* s'inclou al cos de la petició:

```
req.open("POST", url + "autors");
//Informa al servidor que les dades arriben en format JSON
req.setRequestHeader("Content-Type", "application/json");
req.send('{"nomAut":"DIEZ RODRIGUEZ,MIQUELA",
            "dnaixAut":null,"imgAut":null, "fkNacionalitat":null}');
```

Altres mètodes menys utilitzats: HEAD, OPTIONS, TRACE, CONNECT, ...

És diu que GET és un mètode segur perquè no altera l'estat del servidor.

És diu que GET, PUT i DELETE són idempotents perquè no importa quantes vegades es repeteixi la mateixa petició, l'estat del servidor no canvia. En canvi POST no ho és.

Evidentment és el programador el que ha d'utilitzar aquests mètodes de forma coherent, res impedeix programar el PUT per fer un DELETE de la base de dades.

Headers

Quan ens comunicam amb una API REST les capçaleres de les peticions i de les respostes tenen molta importància, ja que determinen, per exemple, el format de dades que espera rebre el client o el que enviam al servidor.

Per exemple al servidor pot ser que només esperin rebre dades amb json, per tant si les enviam amb xml tendrem un problema, i al cas contrari també, si el client espera xml i el servidor envia json el client no serà capaç d'interpretar les dades que li han arribat.

Algunes capçaleres

- **Accept**: El tipus de dades que accepta el client com a resposta. Si el servidor no li pot donar les dades amb aquest tipus enviarà un estat ***406 NotAcceptable***.
- **Accept-Language**: El client envia els codis dels idiomes per ordre de preferència en els que vol rebre el contingut de la resposta.
- **Content-Type**: El tipus de les dades que conté el cos de la petició. Si el servidor no l'accepta tornarà un error ***415 Unsupported Media Type***.
- **Authorization**: Conté dades com l'usauri i el password o un token de seguretat.

Si voleu una llista més exhaustiva de les capçaleres HTTP podeu visitar el següent enllaç: https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers

Http Status Codes

El servidor pot respondre a la petició amb distints codis d'estat, cada un amb el seu significat. Al client, aquests codis ens serveixen per saber que ha passat i respondre correctament a cada situació.

Es divideixen en els següents grups:

- Codis d'informació (100–199)
- Missatges d'èxit (200–299): Tots aquests codis indiquen que la petició s'ha respost correctament, però amb distintes variants, per exemple que no hi ha resposta.
- Redireccions (300–399)
- Errors del client (400–499), per exemple la petició no s'ha creat correctament.
- Errors del servidor (500–599): per exemple que el servidor no sap com satisfer la petició, o que el codi que respon a la petició ha donat una errada.

Alguns codis d'estat

- **200 OK**: La petició és correcta i la resposta també.
- **201 Created**: Com a resposta a la petició s'ha creat un recurs al servidor. Sol ser la resposta als post per fer inserts a la base de dades.
- **204 No content**: La petició s'ha respost correctament però la resposta no conté informació. Per exemple, demanam un alumne amb un identificador que no existeix a la base de dades.
- **301 Moved Permanently**: La petició i totes les següents s'han de redirigir a la url donada.
- **302 Found**: El recurs ha canviat temporalment. S'ha de demanar la nova url, però les següents peticions s'han de dirigir a la url original.

- **400 Bad request**: La petició no s'ha creat correctament i el servidor no la pot entendre.
- **401 Unathorized**: El client s'ha d'autenticar abans de demanar aquest recurs.
- **403 Forbidden**: El client s'ha autenticat però no té permís per fer aquesta petició.
- **404 Not Found**: No s'ha trobat el recurs demanat. No hi ha cap recurs al servidor que respongui a la url demanada.
- **405 Method not allowed**: Estam demanant una operació sobre una url que el servidor no permet, no té implementada. Per exemple volem fer *delete* i el servidor no està preparat per respondre al *delete* a aquella url.
- **406 Not Acceptable***:* El client demana les dades en un format que el servidor no pot servir. Per exemple el client vol les dades en xml i el servidor només les pot tornar en json.
- **415 Unsupported Media Type**: El client envia les dades en un format que el servidor no accepta. Per exemple, el cos de la petició conté dades en JSON i el servidor només les accepta en xml.
- **500 Internal Server Error**: El servidor s'ha trobat en una situació que no sap com resoldre.
- **503 Service Unavailable**: El servidor no està preparat per respondre, perquè ha caigut o perquè té massa peticions simultànies.

Per a més informació podeu consultar els següents enllaços:

- https://developer.mozilla.org/en-US/docs/Web/HTTP/Status
- https://en.wikipedia.org/wiki/List_of_HTTP_status_codes
- http://www.iana.org/assignments/http-status-codes/http-status-codes.xhtml