Software Distribuït - T6 - WEB-Services

Eloi Puertas i Prats

Universitat de Barcelona
Grau en Enginyeria Informàtica

17 d'abril de 2024

Resum de paradigmes de programació distribuïda:

- **Message Passing App.** Aplicació de processos distribuïts que intercanvien dades (ex OpenMPI)
- **Client-Server App.** Aplicació client-servidor amb un protocol prefixat.(ex. mail, ftp, web-server)
- **Web Application.** Aplicació client-servidor amb arquitectura MVC sobre HTTP. Existeixen centernars de Frameworks.
- **RMI/RPC/CORBA App.** Aplicació Distribuïda basada en invocació a mètodes remots.
- **Web Services**. Aplicació Distribuïda basada en serveis sobre HTTP. Existeixen molts Frameworks propis, a part de poder adaptar els de Web Applications.
- **P2P.** Aplicació Distribuïda amb arquitectura Peer2Peer. (ex BitCoins, BitTorrent,...) Existeixen pocs frameworks, la majoria ofereixen infrastructura per a desenvolupar P2P (Tapestry, Pastry, JXTA)

Web Services

- Proveeixen una interficie de servei, equivalent a Objectes Distribuïts, sobre WEB.
- Proporcionen interoperatibilitat per a tota la Internet. Són especialment útils en el camp de la integració B2B (business-to-business).
- S'usen com a middleware per al Cloud Computing.
- Els Web Services no estan pensats per a que siguin accedits pels clients directament via un navegador, sino que siguin aplicacions clients dedicades les que ho facin.

Web Services

- A l'hora de crear un WebService s'ha d'especificar:
  - Representació dades. (JSON,XML,SOAP)
  - descripció dels serveis. (Web Service Description Language (WDSL), Services API)
  - trobar els serveis(Directori de Serveis UDDI, publicar APIS...)
  - accedir als serveis.(HTTP, TCP, SMTP..)

Arquitectures Web Services

- XML-RPC
- SOAP + WSDL
- RESTful. (Representational State Transfer):

RESTful Web Services

- REST (Representational State Transfer) és una aproximació als WebServices amb un estil d'operacions molt restrictiu.
- Els clients usen URL i els mètodes HTTP GET, PUT, DELETE i POST per manipular els recursos representats en XML o JSON.
- L'Èmfasi es posa en la manipulació de les dades, quan un nou recurs es creat (POST), te una nova URL la qual pot ser accedida (GET) o modificada (PUT).

RESTful Web Services

els 4 punts bàsics són:

- Usar els mètodes HTTP **explícitament**.
- Ésser independent d'estats (Stateless)
- Proporcionar URI com a estructures de directoris.
- Transferir XML, JavaScript Object Notation (JSON), o tots dos.

Usar els mètodes HTTP explícitament

POST	Per a crear un recurs en el servidor.
GET	Per a recuperar un recurs.
PUT	Per a canviar l'estat d'un recurs o per a actualitzar-lo.
DELETE	per a esborrar o eliminar un recurs.

Métodes HTTP en el request del client

El mètode HTTP en una petició del client és una paraula reservada (en majúscules), que especifica quina operació del servidor, el client desitja fer. Pot ser segur(+) o no(-) i idempotent(+) o no(-).

+ + GET: per a recuperar el **contingut** de l'objecte web al que fa referència la URI especificada.
+ + HEAD: per a recuperar tan sols la **capçalera** d'un objecte web des del servidor, no el propi objecte.
- - POST: utilitzat per a **enviar dades** a un procés del servidor web.
- + PUT: s'utilitza per demanar al servidor **emmagatzemar** el contingut que s'adjunta amb la petició, a la ubicació del fitxer especificat per la URI en el servidor.

i també però menys utilitzats: DELETE, OPTIONS, TRACE, CONNECT

UNIVERSITAT DE BARCELONA

Exemple Obtenir recurs

```
GET /users/Robert HTTP/1.1
Host: myserver
Accept: application/xml
```

Diferència PUT i POST

```
POST /users HTTP/1.1
Host: myserver
Content-Type: application/xml
<?xml version="1.0"?>
<user>
  <name>Robert</name>
</user>
```

Afegeix Usuari Robert a /users. POST No és idempotent, podries anar afegint varis usuaris si exectues vàries vegades el mateix.

Diferència PUT i POST

```
PUT /users/Robert HTTP/1.1
Host: myserver
Content-Type: application/xml
<?xml version="1.0"?>
<user>
  <name>Bob</name>
</user>
```

Accedeixes a users/Robert i canvies el nom, ara és Bob, per tant el recurs serà users/Bob.

Ésser independent d'estats (Stateless)

Dissenyar l'aplicació fent el servidor stateless:

- Servidor
  - Genera respostes que inclouen referències cap a altres recursos.
  - Genera respostes que indiquen quan la informació pot ser guardada en caché o no, via HTTP els tags Cache-Control i Last-Modified tags en la capçalera de la resposta.
- Client
  - Fa servir el `Cache-control` rebut a la resposta, per tal de fer servir o no la caché que disposi del recurs.
  - Fa servir Conditional GET, mitjançant el tag `If-Modified-Since` per preguntar al server si el recurs ha canviat.
  - Envia peticions complets que puguin ser servits independentment d'altres recursos. Ha de ser independent de sessions en el servidor.

Sistema de Caché: Quan reutilitzar recursos

El servidor posa el header **Cache-Control** en el response per activar el sistema de cache. Així, el client sap que no necessita demanar la pàgina al servidor i pot usar una còpia local.
Paràmetres de Cache-Control:

- **max-age**: És el període de temps en el qual el recurs pot estar guardat en caché (en segons).

  ```
  Cache-Control:public, max-age=31536000
  ```

- **Expires:** El servidor avisa al client a partir de quina data haurà de tornar a demanar el recurs, ja que no serà valid[1].

  ```
  Cache-Control:public Expires:  Mon, 25 Jun 2012
  21:31:12 GMT
  ```

[1] Si tant Expires com max-age estan definits, max-age tindrà preferència.

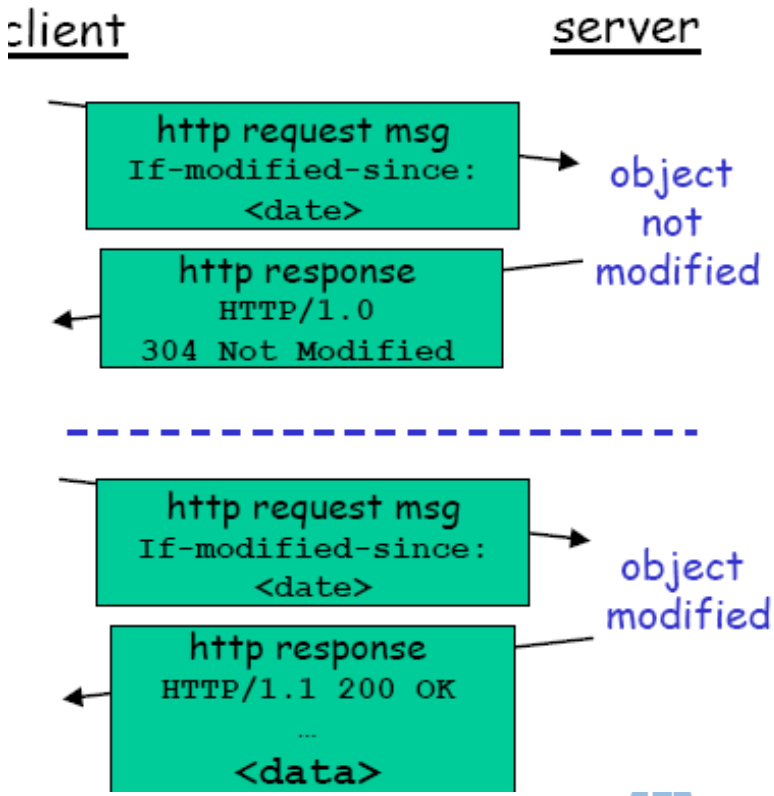
Get Condicional: Com obtenir els recursos

Time-based:

Per activar els requests condicionals, el servidor especifica el temps en que el recurs va ser modificat per últim cop, via el **Last-Modified** de la capçalera del response:

```
Cache-Control:public,
max-age=31536000 Last-Modified:
Mon, 03 Jan 2011 17:45:57 GMT
```

El proper cop que el client demani aquest recurs, obtindrà els seus continguts només si no ha canviat des de la data del *Last-Modified*, via **If-Modified-Since**



Get Condicional: Com obtenir els recursos

Content-based:

- Fa servir l' **ETag** (Entity Tag). Funciona similar al *Last-Modified* excepte que el seu valor és un hash del contingut del recurs.
- Permet al server identificar si el contingut en caché del recurs és diferent a la versió més recent.

```
Cache-Control:public, max-age=31536000 ETag:
"15f0fff99ed5aae4edffdd6496d7131f"
```

- En els següents requests, el client presenta el **If-None-Match** a la capçalera juntament amb el ETag de la última versió obtinguda del recurs.

```
If-None-Match:  "15f0fff99ed5aae4edffdd6496d7131f"
```

- Com el cas anterior, si la versió actual del contingut té el mateix ETag que l'enviat pel client, llavors el servidor envia un codi 304 en comptes del contingut.

Gets Condicionals: Com tractar-los des de Client (exemple)

Part del client: Afegir `If-Modified-Since` en la capçalera seguit del temps `Last-Modified` de la cache

```
request = new XMLHttpRequest();
var ifModifiedSince =
cached.getResponseHeader("Last-Modified") ||
new Date(0); // January 1, 1970
request.open("GET", url, false);
request.setRequestHeader\
    ("If-Modified-Since", ifModifiedSince);
request.send("");
if (request.status == 304) {
        request = cached;
}
else {
        cached = request}
```

Gestionar Caché en el Servidor: Casos d'us

- Pàgines estàtiques: imatges„ CSS, Javascripts. Deixar un any de caducitat:

  ```
  Cache-Control:public; max-age=31536000 Expires:
  Mon, 1 Jun 2017 21:31:12 GMT
  ```

- Pàgines dinàmiques: Depèn de lo dinàmic que pugui arribar a ser podrem posar un Expires raonable.
- Evitar la cache: Recursos segurs o variables sovint requereixen que no hi hagi cache. Per exemple, quan es fa un procés d'adquisició de productes d'una cistella de la compra. En aquests casos s'ha de dir específicament que no volem sistema de cache:

  ```
  Cache-Control:no-cache, no-store
  ```

Gestionar Caché en el Servidor: Implementació

- La majoria de frameworks tenen mecanismes per a tractar la caché de forma automàtica.
- En django es pot fer mitjançant un *decorator* anomenat condition
- No s'ha de confondre amb el Django's cache framework, aquest framework el que fa és guardar en caché pàgines generades dinàmicament per tal de no tornar-les a crear en futures crides de clients. Es tracta d'una optimització en els recursos del servidor i no de la xarxa com en el cas anterior.

Proporcionar URI com a estructures de directoris

- URIs han de ser intuitives, directes, predictibles i fàcilment entendibles.
  - `http://www.myservice.org/discussion/topics/topic`
  - `http://www.myservice.org/discussion/year/day/month/topic`
- Manteniu els noms de les URI's sempre que sigui possible! (campusvirtual2 és un mal exemple)
- Eviteu els query strings el màxim possible.

Transferir XML, JavaScript Object Notation (JSON), o tots dos.

- La representació d'un recurs reflecteix el seu estat actual i els seus atributs en el moment en que el client el demana.
- Aquesta representació és sol fer mitjançant un document estructurat XML o mitjançant un JavaScript Object notation (JSON)
- Existeixen llibreries en la majoria de llenguatges de programació per tal de passar d'objectes o estructures pròpies a estructures equivalents en XML o JSON. (GSON: JAVA-JSON)
- El Content-Type de la resposta s'ha d'adaptar al tipus de document de tornada.
  - JSON: application/json
  - XML: application/xml

Exemple WebService Restful

URL servei
http://date.jsontest.com
Resposta JSON:


```
{
      "time": "03:53:25 AM",
      "milliseconds\_since\_epoch": 1362196405309,
      "date": "03-02-2013"
}
```

Frameworks usats per crear Web Services

- BackEnd específic: Restlets, FastAPI.
- BackEnd WebApps: Ruby On Rails, Django, Servlets ...
- Exemple FullStack: JScript & JQuery -> JSON -> Servlets+GSON

Limitacions del protocol HTTP

1. HTTP és un protocol sense estat, és a dir, el protocol no manté cap informació entre difrents peticions.

   ... però normalment, en una aplicació WEB un usuari fa peticions consecutivament al servidor i es va mantenint la informació fins que es desconecta (manté una *sessió*). **Com es fa?**

2. HTTP és un protocol en que el servidor no pot iniciar una petició a un usuari (no hi ha callback!)

   ... però actualment, els navegadors són capaços de ser notificats quan passa un event a banda del servidor. (existeix *asincronia*) **Com es fa?**

Sessions WEB

- Per a mantenir una sessió d'usuari entre diferents peticions existeixen diferents mecanismes. La idea bàsica, però sempre és la mateixa, el client (navegador) ha d'anar mantenint un "rastre" de l'usuari o de la seva interacció.
- La peça d'informació que vagi deixant dependrà de com dissenyem la web, però el més aconsellable és que el client dugui un "identificador" de sessió i el servidor mantingui les dades relatives a l'usuari per aquella sessió.
- Estratègies:
  - Camps Ocults en el Formulari
  - Cookies
  - URL Rewriting
  - Autentificació via servidor.

Camps ocults

Un camp ocult de formulari és un element **INPUT** en el formulari especificat amb el **TYPE=HIDDEN**.
Contràriament als altres elements **INPUT**, un camp ocult no es mostra en el navegador, però el seu nom i valor es passen igualment en la request query. El nom i valor es generen dinàmicament en el servidor.

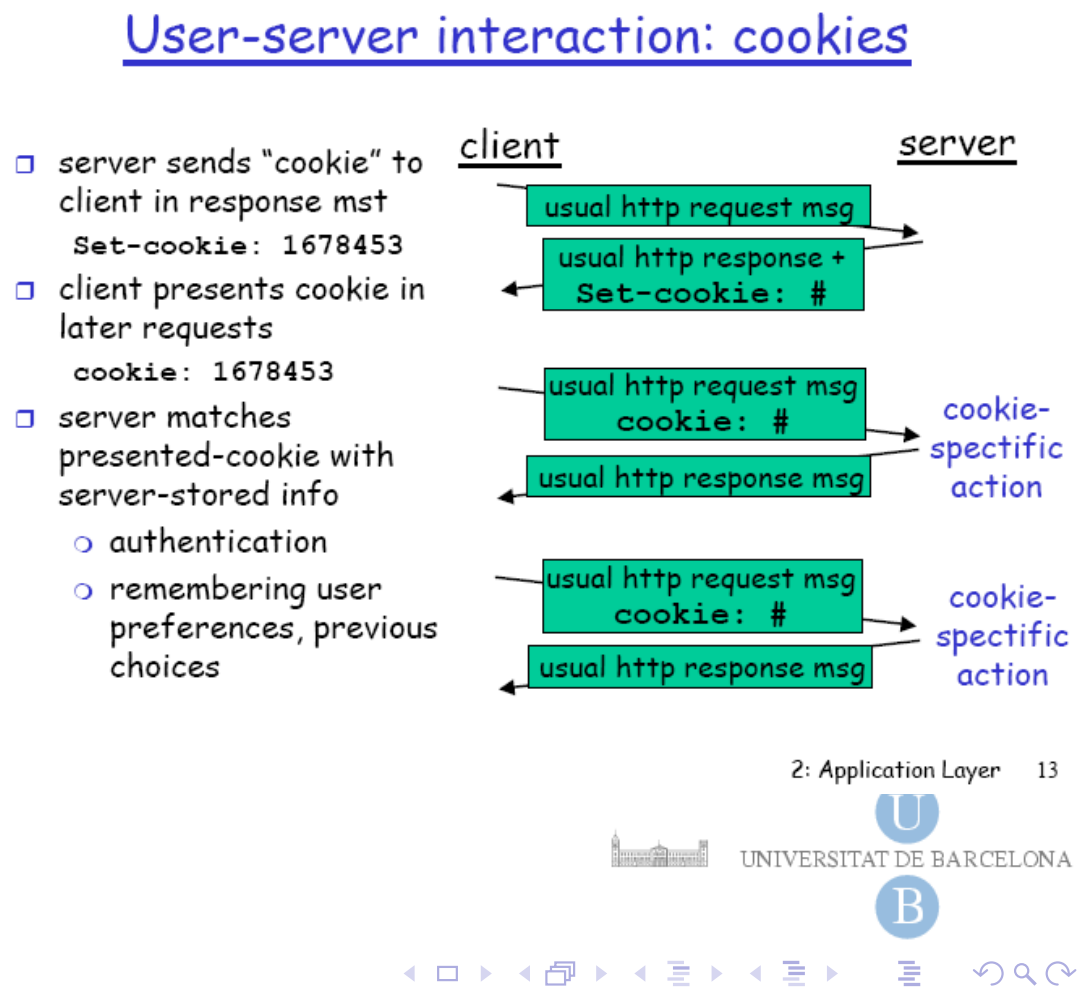
```
<input type=hidden name=id value="12345">
```

Pegues:

- Sistema rudimentari. La informació de sessió s'envia de la mateixa forma que el request query.
- La transmissió de la informació de sessió d'aquesta manera és insegura. Tot i que en el navegador no ho mostri, mirant el codi font de l'HTML des del navegador els camps ocults són perfectament visibles.
- Per tant mai haurien de fer-se servir per a transmetre informació compromesa.

Cookies

• Un esquema més sofisticat per a mantenir informació del servidor per part del client són les galetes o *"cookies"*.
• Fa ús d'una extensió bàsica de l'HTTP que permet una resposta del servidor que contingui una peça d'informació que el navegador pugui emmagatzemar.
• La cookie està associada a un conjunt de URLs per la qual és vàlida. Qualsevol petició futura que faci el client a una d'aquestes URL hi haurà d'associar el valor que disposi en aquell moment de la cookie com a un valor de capçalera.



URL Rewriting

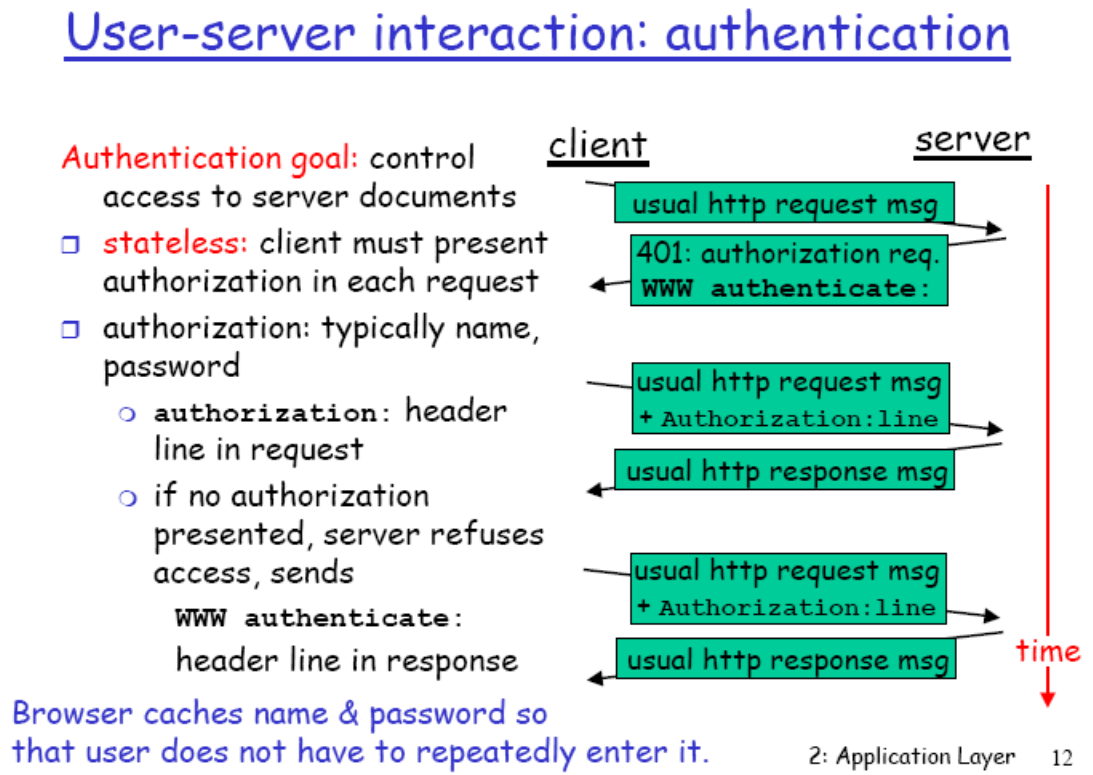
- Les cookies també tenen riscos de seguretat ja que es guarden a banda del client i poden ser manipulades. En la petició i resposta s'estan enviant per la linia de capçalera però igualment no estan codificades
- A més a més, un navegador pot refusar a guardar-se les cookies enviades pel servidor, i per tant fer inviable pel servidor seguir la sessió de l'usuari mitjçant cookies.
- **URL Rewriting** és una solució per passar identificadors de sessió sense necessitat de cookies ni Hidden forms gestionada pel servidor.
- El servidor genera un identificador de sessió molt llarg el primer cop que un usuari es conecta amb ell. Llavors per tot link que apareixi en la resposta es genera una URL amb l'identificador com a paràmetre.
- Si el client accedeix a una URL sense l'identificador perd la sessió.

Aspectes de seguretat en sessions

- Tots els sistemes abans mencionats per intercanviar-se dades de sessió tenen problemes de seguretat ja sigui perquè és fàcil de manipular des de la banda de client, fent factible impersonificació, com pel fet que les dades s'envien en clar, fent factible l'apropiació de dades compromeses per algú que observi el tràfic de la xarxa.
- Les precaucions principals són
  - No enviar cap mena d'informació sensible per a mantenir la sessió, només identificadors generats pel servidor.
  - Fer que aquests identificadors expirin, és a dir, posar un temps *raonable* de caducitat de sessió.

Autentificació d'Accés Bàsic en HTTP

En quant a seguretat d'accés a pàgines, HTTP ens ofereix un sistema d'autentificació, mitjançant el qual en entrar en certes URL específiques se'ns demani d'introduir credencial d'usuari i paraula clau.



Autentificació d'Accés Bàsic en HTTP

- Aabans de la transmissió, l'usuari i la paraula de pas introduïdes es concatenen posant-hi *":"* en mig.
- La cadena resultant és codificada usant l'algorisme Base64.

Per exemple:

`Aladdin:open sesame` $\Rightarrow$ `QWxhZGRpbjpvcGVuIHNlc2FtZQ==`

Les credencials queden ocultes amb l'algorisme Base64, però és trivial descodificar-les. La intenció és, més aviat la d'evitar enviar símbols estranys que puguin haver-hi en la paraula de pas.

La confidencialitat de dades només es pot garantir amb el protocol **HTTPS** que encripta les peticions i resposta amb un sistema de clau pública.

Altres sistemes d'autentificació

- Es pot deixar l'autentificació d'accés a pàgines responsabilitat de l'aplicació web, comprovant les credencials a base de dades.
- Alguns frameworks tenen el seu propi sistema per guardar les credencials, per exemple en el cas de JAVA, es pot fer servir l'especificació JAAS de J2EE.
- Tot i així cal tenir en compte que les creedencials viatjaran en clar per la xarxa, sent sempre necessari fer servir **HTTPS** per a garantir la confidencialitat.

Limitacions del protocol HTTP

1. HTTP és un protocol sense estat, és a dir, a banda de servidor no es manté cap informació entre difrents peticions.

   ... però normalment, en una aplicació WEB un usuari fa peticions consecutivament al servidor i es va mantenint la informació fins que es desconecta (manté una *sessió*). **Com es fa?**

2. HTTP és un protocol en que el servidor no pot iniciar una petició a un usuari (no hi ha callback!)

   ... però actualment, els navegadors són capaços de ser notificats quan passa un event a banda del servidor. (existeix *asincronia*) **Com es fa?**

AJAX (Asynchronous JavaScript and XML)

- Ens permet fer codi client-side amb recepció no bloquejant.
- **XMLHttpRequest**: Objecte JavaScript que envia peticions HTTP.
  - Inicia la petició:
    - Associa una funció anònima que atendrà la resposta a l'event `onreadystatechange` de la petició.
    - Inicia una petició GET o POST
    - Envia les dades
  - Atén la resposta
    - Comprova que l'estat del `readyState` del request sigui 4 i el codi de resposta 200.
    - Extreu la resposta.
    - Processa la resposta.
- JavaScript no es multi-thread. La implementació del navegador és la que s'encarrega de canviar l'estat del request i col·locar un nou esdeveniment en la cua d'events del JavaScript.

WebSockets

- Canal bidereccional i full-duplex sobre un únic socket TCP.
- S'usa entre web browsers i web servers fent possible que hi hagi més intreacció entre ells, facilitant el desenvolupament de jocs i aplicacions amb continguts en viu.
- WebSocket usa un protocol TCP propi. RFC any 2011.
- Només el handshake es realitza usant el protocol HTTP enviant un upgrade request.
- Tota la interacció es fa mitjançant el port 80.

Llibreries i Frameworks amb AJAX

Escriure funcions Javascript compatibles amb tots els navegadors des de zero és molt complicat. Existeixen llibreries i Framewoks per ajudar a fer el frontend d'una pàgina web generant el JavaScript que executarà navegador. S'encarreguen de la part d'AJAX, però també de modificar el DOM de l'HTML, les fulles d'estil CSS i fins i tot la navegació entre pàgines i gestió de credencials.

- Llibreries com JQuery, React.js
- Frameworks fullstack com Vue.js o Angular

Entorn de Desenvolupament JavaScript.

- Els navegadors porten eines d'ajuda al desenvolupament (Firefox, Chrome, Safari). Entre d'altres hi trobem:
  - Consola de Javascript
  - Web Inspector
  - Codi
  - Xarxa...
- aquestes eines són indespensables per a poder debugar i testejar el nostre codi.