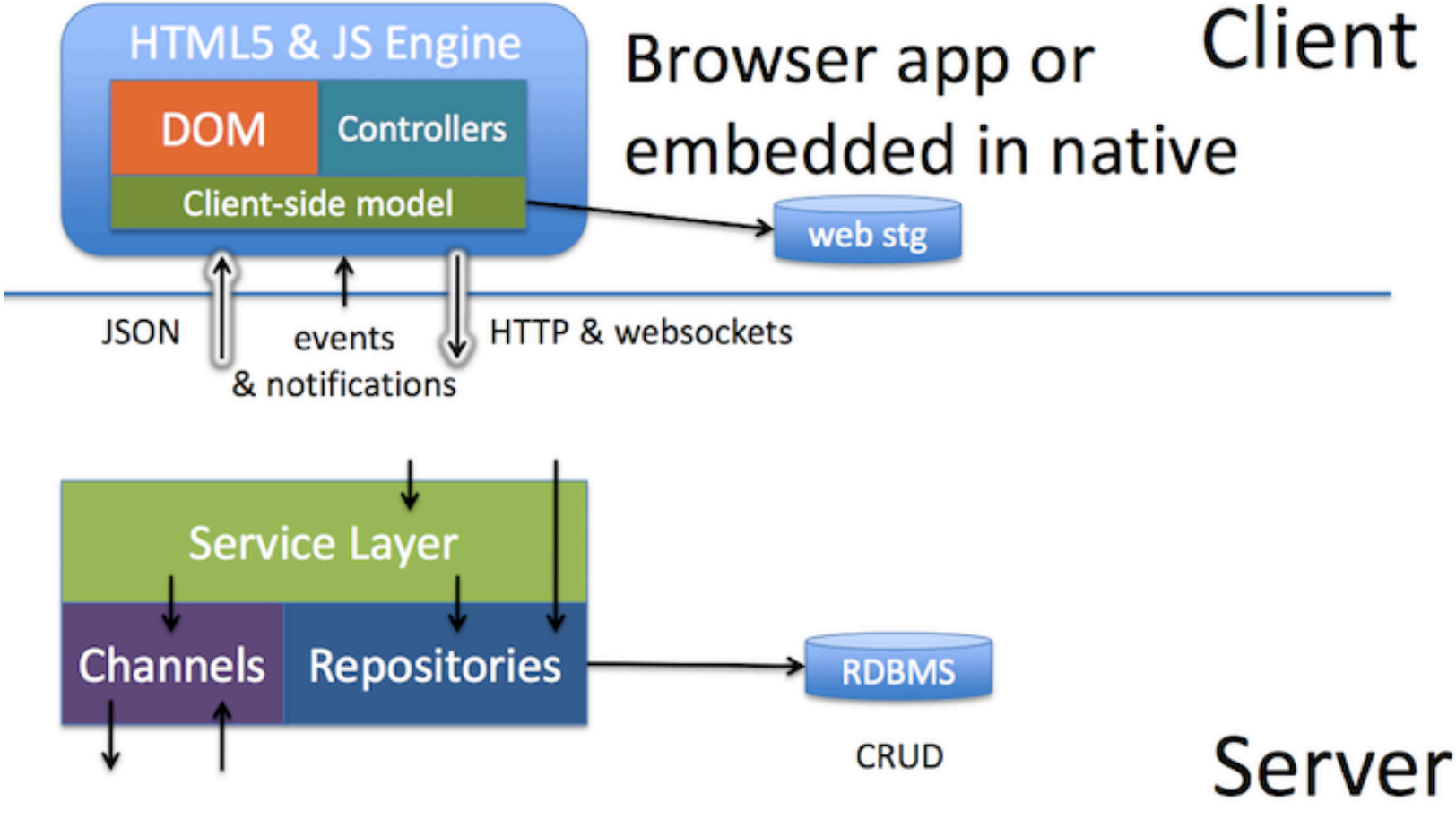
Tema 4: Javascript en clientes web

Parte II: Comunicación con el servidor y persistencia local

Necesitamos alguna funcionalidad de JS en el navegador para poder hacer peticiones al servidor (Un API, típicamente REST, salvo casos como Firebase)



- El servidor actúa como una especie de "repositorio remoto de datos"
- Además nos vendría bien tener un repositorio local

4.4. AJAX

AJAX

Asynchronous Javascript And XML

Es una combinación de tecnologías:

- **API `fetch`** para **hacer peticiones HTTP al servidor** con Javascript y recibir la respuesta sin cambiar de página
- ***XML***: al comienzo era el formato de intercambio de datos cliente/servidor. Hoy en día se usa JSON.
- **API DOM**: ya visto, permite actualizar **solo parte de la página** con los datos del servivor

Formato de datos en AJAX

Originalmente se usaba XML para intercambiar datos, pero es tedioso de *parsear* con JS (hay que usar el API DOM), mientras que convertir una cadena de texto JSON a objeto JS es trivial con el API estándar

```
//Si esto fuera AJAX de verdad, este texto vendría en la respuesta del servidor
var texto = '{"login":"pepe", "nombre": "Pepe Pérez"}'
//De cadena JSON a objeto JS
var objeto = JSON.parse(texto)
console.log(objeto.login) //pepe
//De objeto JS a cadena JSON
console.log(JSON.stringify(objeto))
```

Hacer peticiones HTTP con el `fetch` API

```
<img src="" id="avatar">
```

```
try {
  var respuesta = await fetch('https://api.github.com/users/octocat')
  //la conversión de cadena JSON a texto es asíncrona!!!
  var datos = await respuesta.json()
  document.getElementById("avatar").src = datos.avatar_url
  if (respuesta.ok) {
      console.log('El servidor devuelve OK: ' + respuesta.status)
  }
}
catch (error) {
    console.log('FAIL!!')
    console.log(error)
}
```

https://jsbin.com/sarahuc/edit?html,js,output

Versión con promesas

```
fetch('https://api.github.com/users/octocat')
  .then(function(respuesta){
     return respuesta.json()
  })
  .then(function(datos){
    console.log(datos)
     document.getElementById("img_avatar").src = datos.avatar_url
  })
  .catch(function(error){
      console.log('FAIL!!')
      console.log(error)
  })
//Cuidado, fetch es asíncrono
console.log('Cuando se ejecuta esto todavía no se ha recibido la respuesta!!')
```

https://jsbin.com/yuhutiz/edit?html,js

Peticiones más complejas con `fetch`

Por defecto se hace una petición **GET**. Para cambiar el tipo de petición, añadir cabeceras, cuerpo de petición, etc, podemos pasar un segundo parámetro que es un objeto JS con las propiedades:

https://jsbin.com/razelec/edit?html,js,console

```
//reqres.in es un API REST "fake" al que podemos hacer peticiones
var usuario = {};
usuario.login = "Pepe"
usuario.nombre = "Pepe Pérez"
var respuesta = await fetch('https://reqres.in/api/users', {
    method: 'POST',
    headers: {
      'Content-type':'application/json'
    },
    body: JSON.stringify(usuario)
})
var datos = await respuesta.json();
console.log(datos.login)
```

Formularios y AJAX

En **aplicaciones "tradicionales"** el navegador es el responsable de enviar los datos tecleados en los campos del formulario, y este proceso se desencadena automáticamente con un `input` de `type=submit`.

En **aplicaciones con AJAX** y formularios el código JS es el que debe recolectar los datos contenidos en los campos y enviarlos con `fetch`. Este proceso se puede desencadenar con un `<button>` convencional, no hace falta un . Si usamos un `type=submit` necesitaremos `preventDefault()` para que el navegador no desencadene el envío, *en caso contrario la página actual se perdería*.

```
<input type="text" id="login"/>
<input type="password" id="password"/>
<!-- Fijaos en que el botón no es submit -->
<button id="boton">Dar de alta</button>
```

El JS lanza un `fetch` cuando se pulsa el botón

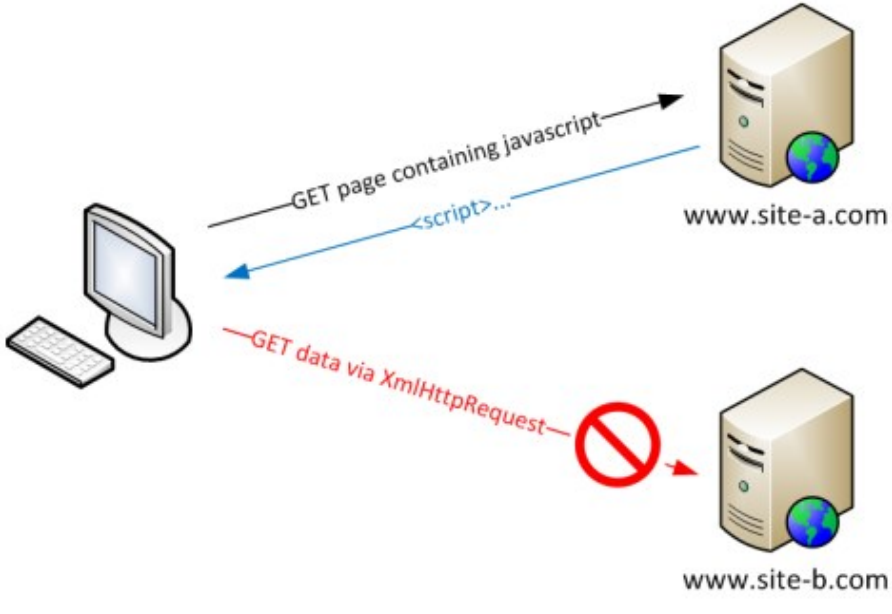
```
document.getElementById('boton').addEventListener('click', function(){
  var datos = {
       login: document.getElementById('login').value,
       password: document.getElementById('password').value
      }
  fetch('http://reqres.in/api/users', {
    method: 'POST',
    headers: {
      'Content-type':'application/json'
    },
    body: JSON.stringify(datos)
  })
})
```

4.5 CORS

Restricciones de seguridad

Política de seguridad del "mismo origen": un `fetch` solo puede hacer una petición al mismo *host* del que vino la página en la que está definido

Por ejemplo, el Javascript de una página de `www.vuestrositio.com` en principio no puede hacer peticiones AJAX a Facebook (salvo que FB lo permita **explícitamente**)



CORS

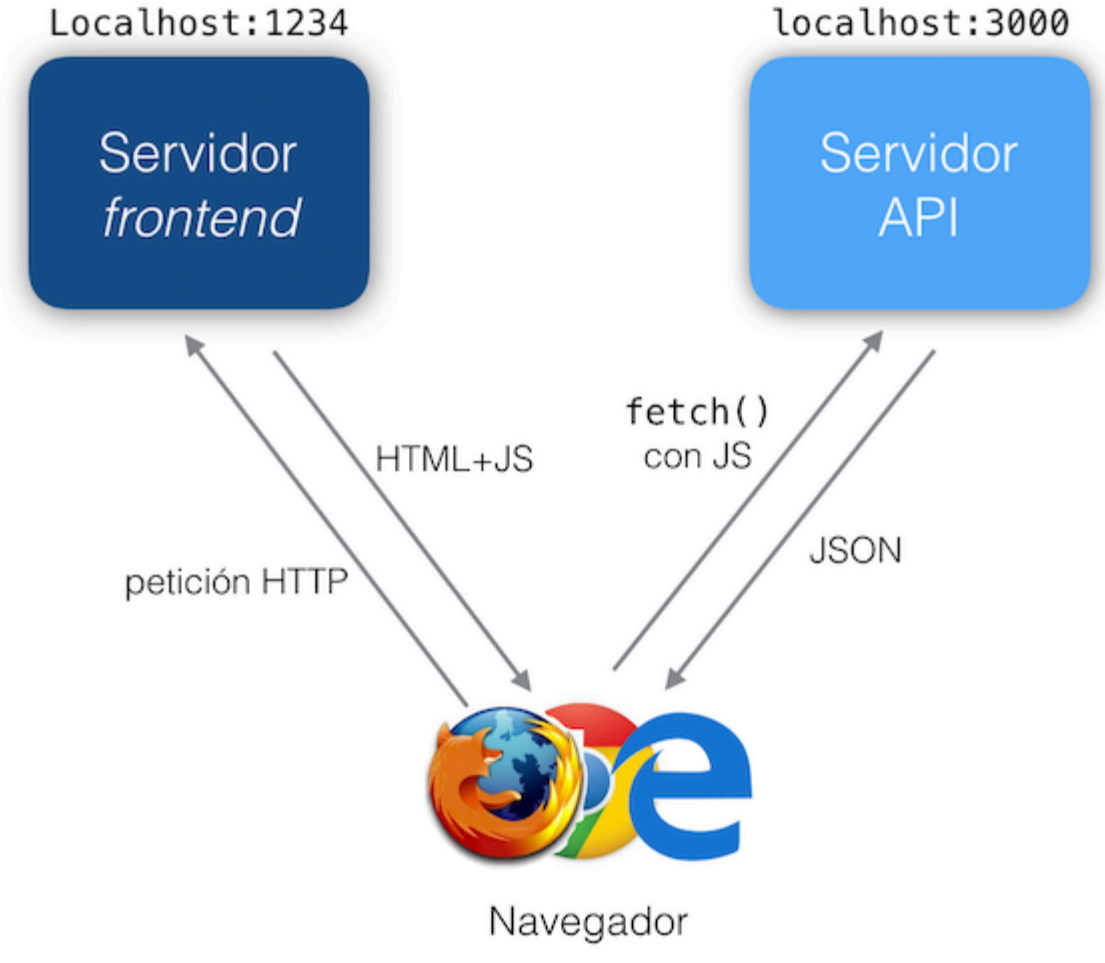
(Cross Origin Resource Sharing) : permite saltarse la *same origin policy* con la colaboración del servidor

- En cada petición *cross-domain* el navegador envía una cabecera `Origin` con el origen de la petición. Es imposible falsearla desde JS
- El servidor puede enviar una cabecera `Access-Control-Allow-Origin` indicando los orígenes desde los que se puede acceder a la respuesta. Si encajan con el origen de la petición el navegador dará “luz verde”

```
HTTP/1.1 200 OK
Server: Apache/2.0.61
Access-Control-Allow-Origin: *
```

¿Necesitamos CORS para usar un API propio?

No, pero nos da la libertad de que el servidor del API y del sitio web sean distintos



Implementar CORS en el servidor

Aunque "simplemente" se trata de generar las cabeceras adecuadas desde el servidor, en prácticamente todos los lenguajes de programación en el servidor tenemos alguna librería/paquete/plugin... que añade CORS a nuestra aplicación de modo sencillo.

Por ejemplo en Express tenemos el paquete `cors`:

```
var cors = require('cors')
//Suponiendo "app" la variable obtenida como app=express()
app.use(cors())
//cors es un middleware, un software entre la petición y la respuesta
```

Aclaración: CORS **no es un mecanismo de protección del servidor**. Nada nos impide escribir código en Java/Python/Ruby...etc, incluso NodeJS, que pueda hacer la petición HTTP desde fuera del navegador. Se trata de una **limitación de JS en el navegador**

4.6
Persistencia local

¿Por qué almacenar datos en el cliente?

- **Variables compartidas** entre diferentes páginas de la aplicación: recordemos que el ámbito de una variable JS es el HTML en el que está el JS que la define
- **Datos permanentes** que queremos conservar entre diferentes "sesiones" de navegación. Por ejemplo guardar un *token* JWT para no tener que autentificarnos siempre.
- **Datos que no podemos sincronizar** con el servidor por ejemplo por falta de conectividad en este momento (por ejemplo, porque estamos en el metro)

API Local Storage

- Objeto global `localStorage` donde podemos almacenar **pares "clave/valor"** que no se pierden aunque se cierre el navegador
- Podemos almacenar un dato bajo una clave, recuperarlo conociendo la clave o iterar por todas las claves y valores

```
localStorage.login = "pepe"
```

- El valor es siempre una **cadena**, para otro tipo de datos hay que hacer la conversión manualmente
  - Aunque ya hemos visto que la conversión objeto<->cadena es sencilla gracias a `JSON.stringify` y `JSON.parse`
- El **ámbito** es el sitio web actual. Hay otro objeto `sessionStorage` que reduce más este ámbito, a la "pestaña" actual del navegador, y además se pierde si se cierra.

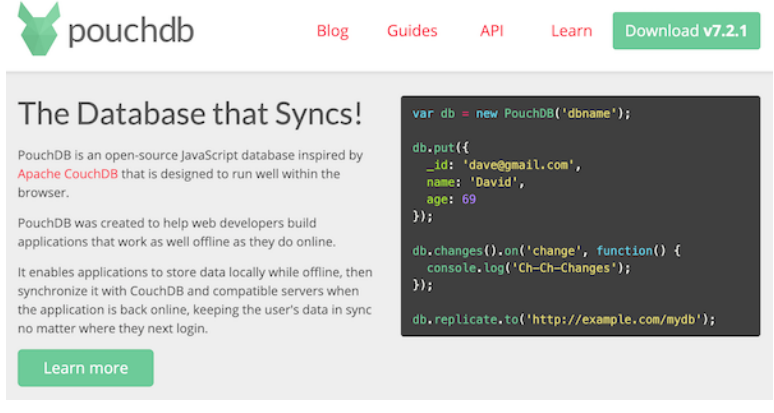
Ejemplo de `localStorage`

```
//http://jsbin.com/bofabe/edit?html,js,output
function guardarNombre() {
  nombre = prompt("¿cómo te llamas?")
  localStorage.setItem("usuario", nombre)
  //esta sintaxis es equivalente a lo anterior
  localStorage.usuario = nombre
  //y esta también
  localStorage["usuario"] = nombre
  edad = prompt("¿Cuántos años tienes?")
  localStorage.edad = edad
}
function mostrarNombre() {
  alert("Me acuerdo de ti, " + localStorage.usuario +
  " vas a cumplir " + (parseInt(localStorage.edad) + 1) + " años!!")
}
function mostrarTodosLosDatos() {
  datos=""
  for(var i=0; i<localStorage.length; i++) {
     clave = localStorage.key(i)
     datos = datos + clave + "=" + localStorage[clave] + '\n'
  }
```

Bases de datos en el cliente

- El API nativo es IndexedDB: una base de datos de pares clave-valor (tipo NoSQL).
- Probablemente es de demasiado bajo nivel como para usarlo directamente

Dexie



PouchDB

Ya veremos esto en los temas de dispositivos móviles