HTML5 APIS

APIS FOR WEBAPPLICATIONS

NEBENLÄUFIGKEIT FÜR JAVASCRIPT

WEBWORKER

BROWSER TABS SIND SINGLE THREADED

- Keine gleichzeitige Ausführung von mehreren parallel arbeitenden Skripten.

WORKER

- nebenläufige Hintergrundskripte über mehrere Threads.
- rechenintensive Aufgaben blockieren nicht den sonstigen Ablauf der Webanwendung.

DAS WORKER PRINZIP

```
//main.js:

var worker = new Worker('extra_work.js');
worker.onmessage = function(event) {
    alert(event.data);
};


//extra_work.js:

self.onmessage = function(event) {
    // Do some work.
    self.postMessage(some_data);
}
```

HTML5 WORKER BROWSER

- Chrome 4+
- Firefox 3.5+
- Safari 4+
- IE 10+
- Opera 11.5+

BROWSERSUPPORT

```
if(typeof(Worker) !== „undefined") {

    // Wenn der Web Worker unterstützt wird,
    // rufe das Skript auf.

} else {

    document.getElementById("random").innerHTML
    =  "Oops, Ihr Browser unterstützt
        keine Worker Objekte.";

}
```

EIN WEBWORKER OBJEKT ANLEGEN

```
if(typeof(myWorker) === „undefined") {

   myWorker = Worker(„random_work.js");

}
```

NACHRICHTENHANDLING ÜBER DIE MESSAGE API

EVENT LISTENING - DATEN VOM WORKER OBJEKT ERHALTEN

```
// Worker:
something.postMessage(m);


// Recipient
myWorker.onmessage = function (event) {

    document
        .getElementById("random")
        .innerHTML = event.data;

};
```

DEN WEB WORKER BEENDEN

```
function stopWorking() {

    // beendet den Web Worker
    myWorker.terminate();

}
```

SHARED WORKERS

- Ein einfacher Worker kann nur von der Seite aufgerufen werden, die ihn erzeugt hat.
- Shared Worker können von mehreren Seiten einer Domain verwendet werden.

HTML5 SHARED WORKER BROWSER

- Chrome 4+
- Firefox 29+
- Safari 5 - 6
- IE --
- Opera 11.5+

EINEN SHARED WORKER ERZEUGEN

```
var worker = new SharedWorker("shared-worker.js");

// Alle Seiten, die sich eine Instanz des Shared Workers
// erstellen, arbeiten im Hintergrund mit demselben Worker.
```

SICH MIT EINEM SHARED WORKER VERBINDEN

```
/* Ein Shared Worker besitzt Ports für die verschiedenen
Seiten, über die er kommuniziert. Diese API ist der HTML5
Messaging API ähnlich. */

var worker = new SharedWorker("/html5/web-worker-shared.jsp");
worker.port.addEventListener("message",
        function(event) {
            alert(event.data);
        }
        , false
);
worker.port.start();
```

EINE NACHRICHT AN DEN SHARED WORKER SCHICKEN

```
/* Wenn der Port gestartet ist und die Seite auf Messages hört,
lassen sich Nachrichten an den Shared Worker schicken */

worker.port.postMessage(
   "Meine Nachricht"
);
```

```
var ports = [] ;
onconnect = function(event) {
    var port = event.ports[0];

ports.push(port);
    port.start();
    port.addEventListener("message",
        function(event) { listenForMessage(event, port); } );
}

listenForMessage = function (event, port) {
    port.postMessage("Reply from SharedWorker to: " + event.data);
}

//Implementation of shared worker thread code
setInterval(function() { runEveryXSeconds() }, 5000);

function runEveryXSeconds() {
    for(i = 0; i < ports.length; i++) {
        ports[i].postMessage("Calling back at : "
                + new Date().getTime());
    }
}
```

ECHTZEITKOMMUNIKATION ZWISCHEN CLIENT UND SERVER

WEBSOCKETS

„Mit AJAX Requests kann ich über dem Browser keine Modelleisenbahn steuern.“

– JAN HICKSON

HTTP IST STATUSLOS

- -> Request, Response, Ende
- Echtzeit ist nicht möglich, da Client und Server bei jedem Request erneut verhandeln, bevor Daten übertragen werden.

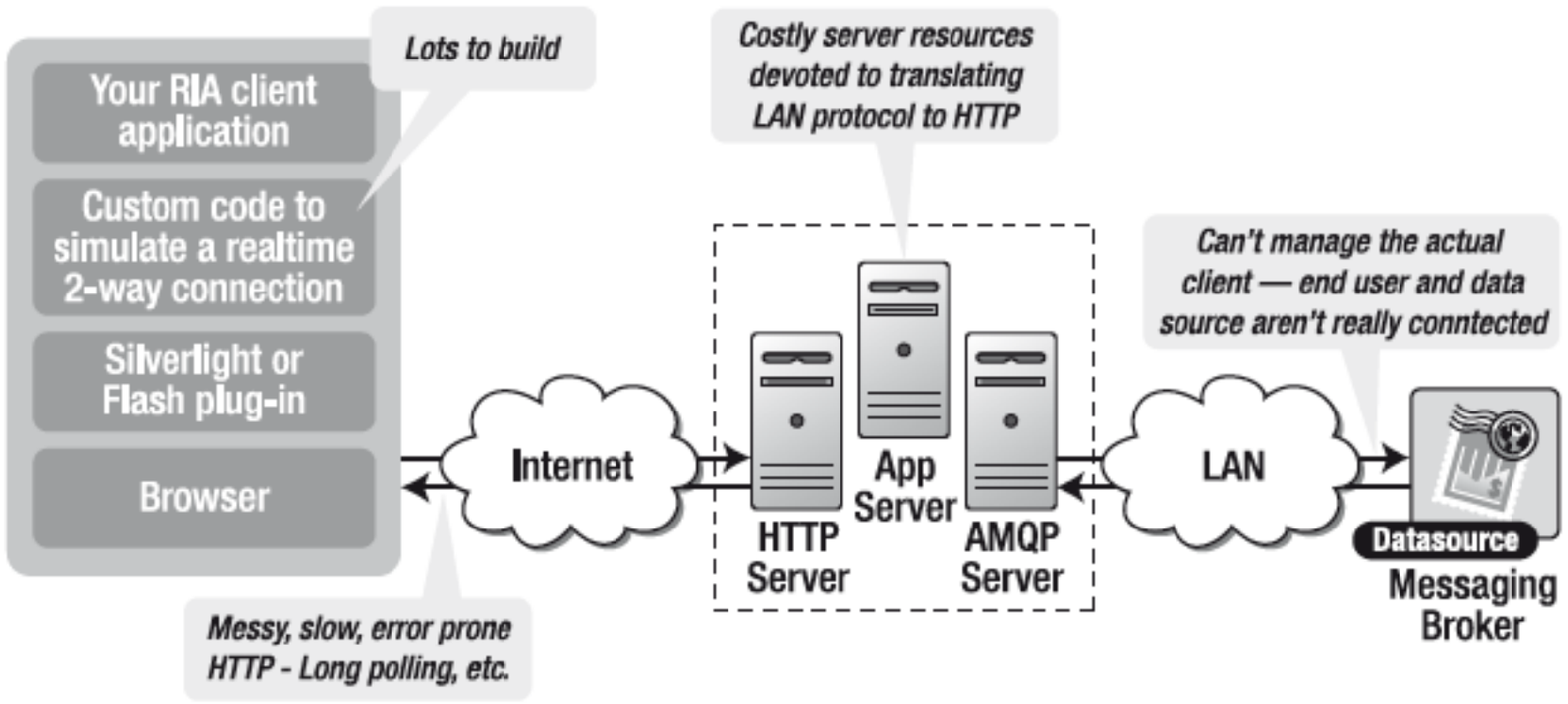
ECHTZEITVERBINDUNGEN?

- Seiten aktualisieren
- Polling
- Longpolling

- Polling und Longpolling setzen dabei in regelmäßigen Intervallen Requests an den Server ab, um neue Informationen zu erhalten.

- Beim Longpolling hält der Server die Verbindung eine Zeit lang offen.

- Keine echte Synchronisation mit severseitigen Informationsupdates



WEBSOCKETS

- ein Protokoll, das eine persistente Verbindung zwischen Browser und Webserver offenhält.

- ws:// - Websocket Protokoll
  wss:// - Websocket Secure Protokoll

BIDIREKTIONAL & FULLDUPLEX

- Über diese Verbindung kann in beiden Richtungen (Client <--> Server) kommuniziert werden.
- Aufgrund des binären Protokolls (ws:/wss:) hat es sehr wenig Overhead.

HTTP INITIALISIERUNG

- Der initiale Verbindungsaufbau einer Websocket-Verbindung läuft über HTTP (oder HTTPS),
- Ein Upgrade-Header teilt dem Server mit, dass auf das Websocket-Protokoll „upgegradet“ werden soll.
-

```
// From client to server:
GET /chat HTTP/1.1
Host: server.example.com
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Key: dGhlIHNhbXBsZSBub25jZQ==
Origin: http://example.com
Sec-WebSocket-Protocol: chat, superchat
Sec-WebSocket-Version: 13

// From server to client:
HTTP/1.1 101 Switching Protocols
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Accept: s3pPLMBiTxaQ9kYGzzhZRbK+xOo=
Sec-WebSocket-Protocol: chat
```

SERVER PROXY

- Serverseitig wird ein Socketproxy installiert, zum Beispiel in node.js.

- nodejs beherrscht HTTP und Websockets und kann einen Upgrade-Headers verarbeiten.

WEITERE SOCKET SERVER

- **Jetty**
  HTTP-Server und Servlet-Container, der WebSockets seit Version 7.0.1 unterstützt.

- **phpwebsocket**
  einer der ersten WebSocket-Server in PHP

- **jWebSocket**
  High-Speed Kommunikationsserver inkl. Web, Java und Mobile Clients, Open Source

- **xLightweb**
  HTTP-Bibliothek inkl. HttpClient und HttpServer, welche WebSocket und ServerSentEvent unterstützt

SOCKET TOOLS

- **Tomcat**
  ab Version 7.0.27

- **PyWebsocket**
  Python Websocket Server (Apache Modul oder Standalone)

- **Node.js**
  Serverseitiges Javascript, mit dem ein Webserver geschrieben werden kann

- **Socket.io**
  Ein Framework, mit dem realtime apps für jeden Browser und jedes mobile Gerät möglich sind.

HTML5 SOCKETS BROWSER

- Chrome 4+
- Firefox 4+
- Safari 5+
- IE 10+
- Opera 11.5+

WAS BRINGEN WEBSOCKETS?

VORTEILE DER WEBSOCKETS GEGENÜBER HTTP REQUESTS

- Geschwindigkeitsteigerung
- Datenmengenreduktion

HEAD EINES HTTP-REQUEST

```
GET /PollingStock//PollingStock HTTP/1.1
Host: localhost:8080
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.9.1.5)
Gecko/20091102
Firefox/3.5.5
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-us
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive
Referer: http://www.example.com/PollingStock/
Cookie: showInheritedConstant=false;
showInheritedProtectedConstant=false;
showInheritedProperty=false; showInheritedProtectedProperty=false;
showInheritedMethod=false; showInheritedProtectedMethod=false;
showInheritedEvent=false; showInheritedStyle=false;
showInheritedEffect=false
```

- 1,000 Clients pollen jede Sekunde
  6,968,000 bits per second
  6.6 Mbps

- 10,000 Clients pollen jede Sekunde
  69,680,000 bits per second
  66 Mbps

- 100,000 Clients pollen jede Sekunde
  696,800,000 bits per second
  665 Mbps

HEAD EINES SOCKET-REQUEST

```
\0x00 Hello, WebSocket \0xff
```

0,002%

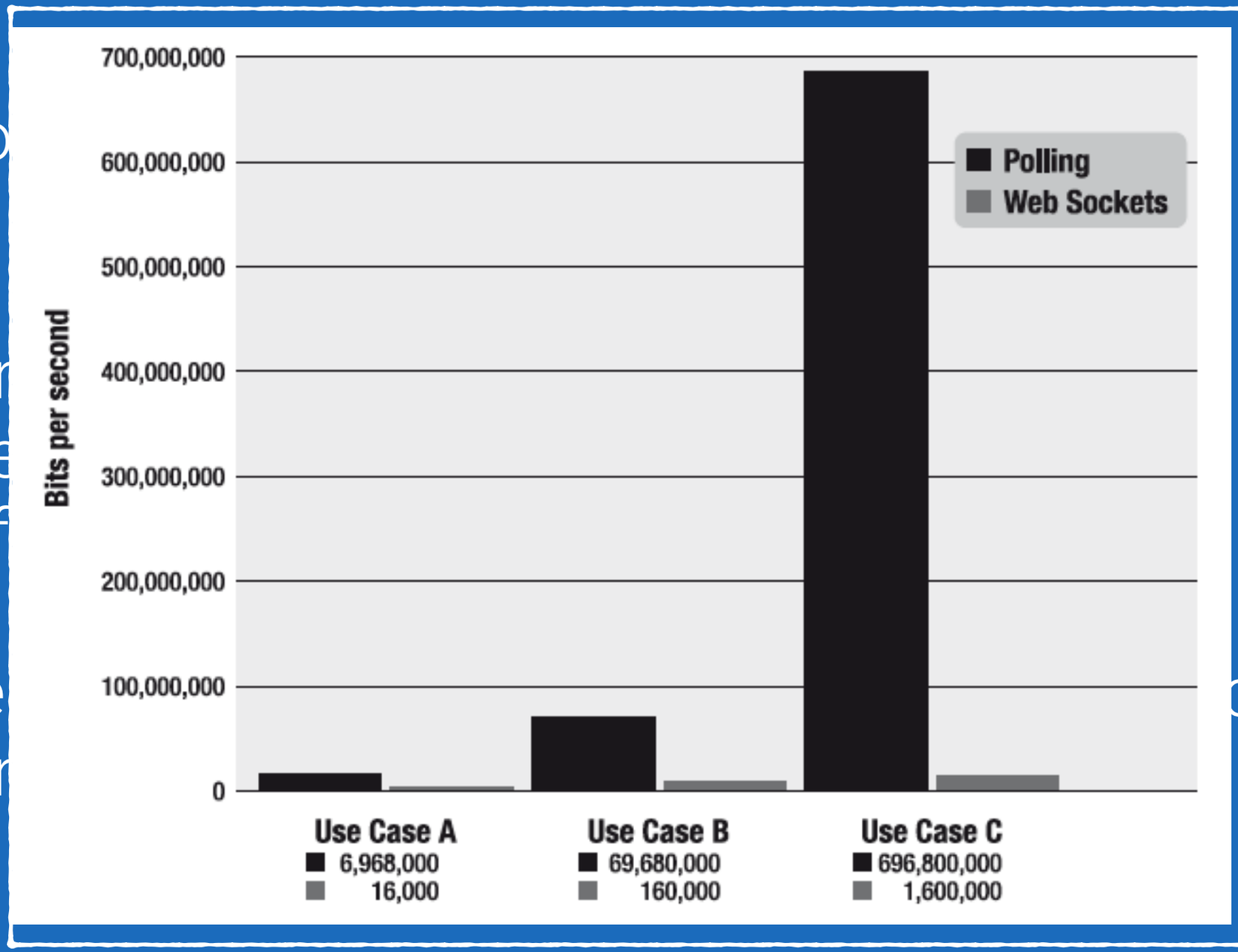
- 1,000 Clients erhalten 1 Nachricht pro Sekunde
  16,000 bits per second
  0,015 Mbps vs. 6.6 Mbps

- 10,000 Clients erhalten 1 Nachricht pro Sekunde
  160,000 bits per second
  0,153 Mbps vs. 66 Mbps ->

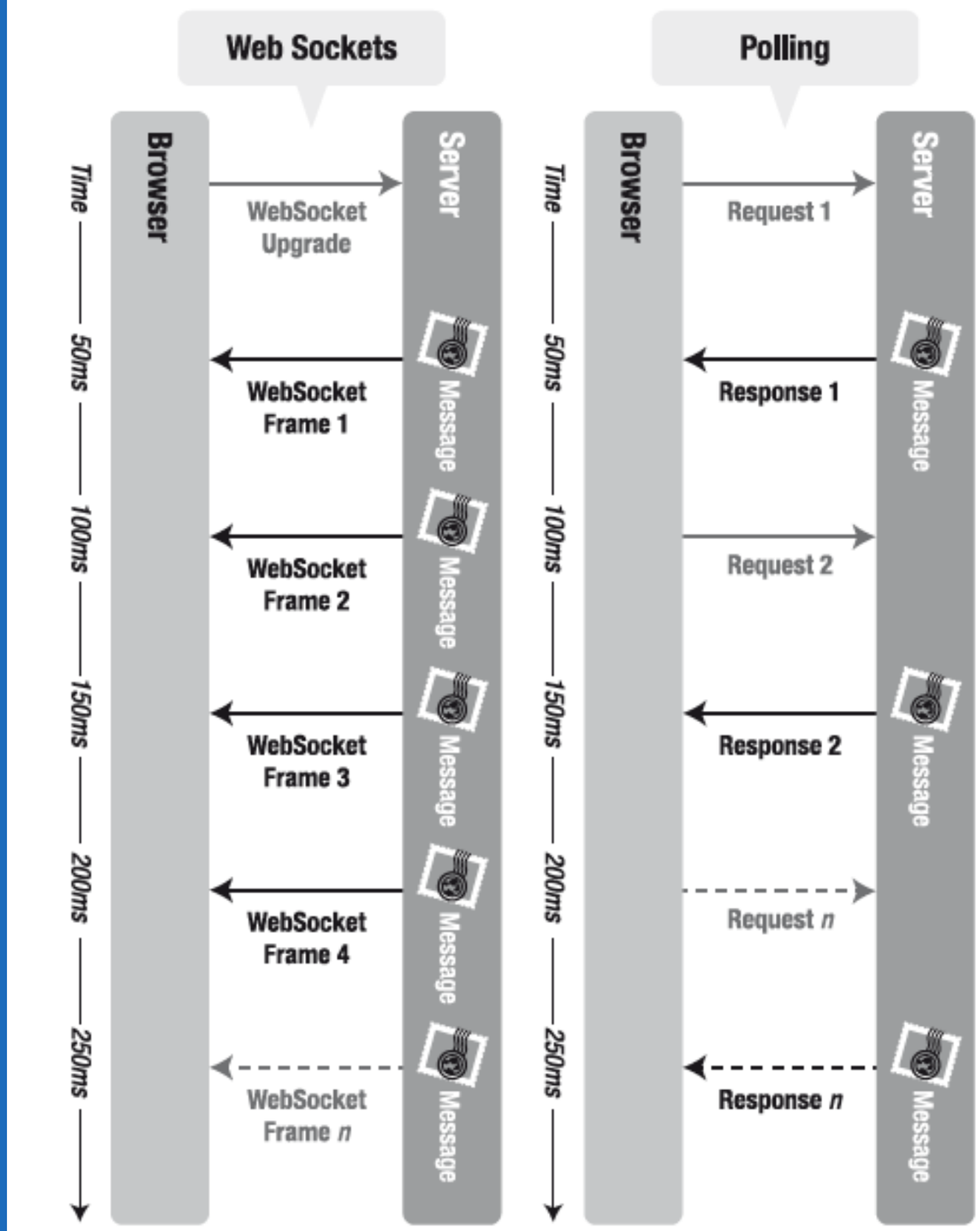
- 100,000 Clients erhalten 1 Nachricht pro Sekunde
  1,600,000 bits per second
  1,526 Mbps vs. 665 Mbps

Datenmengenvergleich des HTTP-Overheads

- Bro
- Chr
  Fire
  Saf
- Die
  Chr



Quelle: Pro HTML 5 - Programming Powerful



DAS WEBSOCKET INTERFACE

```
// The Websocket Interface - start und ready state section

[Constructor(DOMString url, optional (DOMString or DOMString[])
protocols)] interface WebSocket : EventTarget {
  readonly attribute DOMString url;

  // ready state
  const unsigned short CONNECTING = 0;
  const unsigned short OPEN = 1;
  const unsigned short CLOSING = 2;
  const unsigned short CLOSED = 3;

  readonly attribute unsigned short readyState;
  readonly attribute unsigned long bufferedAmount;
...
```

```
// The Websocket Interface – networking

[TreatNonCallableAsNull] attribute Function? onopen;
[TreatNonCallableAsNull] attribute Function? onerror;
[TreatNonCallableAsNull] attribute Function? onclose;

readonly attribute DOMString extensions;
readonly attribute DOMString protocol;

void close([Clamp] optional unsigned short code,
   optional DOMString reason);
...
```

```
//The Websocket Interface – messaging and end section

[TreatNonCallableAsNull]
attribute Function onmessage;
attribute DOMString binaryType;

void send(DOMString data);
void send(ArrayBufferView data);
void send(Blob data);
};
```

Quelle: http://dev.w3.org/html5/websockets/; Hickson

DIE JAVASCRIPT API FÜR DEN CLIENT

DAS READYSTATE ATTRIBUT ENTHÄLT DEN STATUS DER VERBINDUNG

CONNECTING (numeric value 0)
Die Verbindung wurde noch nicht hergestellt.

OPEN (numeric value 1)
Die Verbindung steht, Kommunikation ist möglich.

CLOSING (numeric value 2)
Die Verbindung führt den Closing Handshake aus.

CLOSED (numeric value 3)
Die Verbindung wurde geschlossen oder konnte nicht hergestellt werden.

DIE WEBSOCKET METHODEN

```
mySocket = new WebSocket();

mySocket.onopen = function(evt) {
   console.log(„Connection open ...");
};

mySocket.onmessage = function(evt) {
   console.log( "Received Message: " + evt.data);
};

mySocket.onclose = function(evt) {
   console.log("Connection closed.");
};

mySocket.onerror = function(evt) {
   console.log("An error happened.");
};
```

SOCKET ÖFFNEN UND DATEN SENDEN

```
var mySocket = new WebSocket('ws://game.example.com:12010/
updates');

mySocket.onopen = function () {

   setInterval(function() {

    if (mySocket.bufferedAmount === 0) {
       mySocket.send( );
    }

  }, 50);

};
```

ONMESSAGE

```
mySocket.onmessage = function (event) {

    if (event.data === 'on') {
       turnLampOn();
    } else if (event.data === 'off') {
       turnLampOff();
    }

};
```

JAVASCRIPT AUF DEM SERVER

NODEJS SOCKETS

NODEJS

- node.js ist ein Javascript Framework für den Servereinsatz.
- Es setzt auf der Opensource Javascript Engine V8 auf und wurde 2009 von Ryan Dahl entwickelt.
- Ein Socketproxy in nodejs ist einfach zu schreiben.

- Es läuft unter Windows, MacOs oder Linux. Zur Erweiterung gibt es einen Package Manager (npm).

- Nach der Installation kann node.js in der Konsole/im Terminal ausgeführt werden.

EIN HTTP SERVER IN NODE.JS

```
// server.js

var http = require(„http“);

http.createServer(function (request, response) {

   response.writeHead(200, {'Content-Type': 'text/plain'});
   response.end('Hello World\n');

}).listen(80, '127.0.0.1');

console.log(,http server runs at http://127.0.0.1:1337/');


$ node server
http-server is runs at http://127.0.0.1:80/
```

EIN SOCKETPROXY

```
// Websocket-Server (-> npm install ws!)

var WebSocketServer = require('ws').Server
var wss = new WebSocketServer({
    host: „192.168.2.1“,
    port: 8000
});

wss.on('connection', function(ws) {
    console.log('client verbunden...');

    ws.on('message', function(message) {
        console.log('von Client empfangen: ' + message);
        ws.send('von Server empfangen: ' + message);
     });
});
```

DER BROWSERPART ZUR SOCKETKOMMUNIKATION

```
function connect() {
        // Websocket
        var socket = new WebSocket(„ws://192.168.2.1:8000“);

        socket.onopen = function() {
            console.log(„Socket Status: “
            + socket.readyState + „ (open)“);
        }

        socket.onmessage = function(msg) {
            console.log(„Empfangen: “ + msg.data);
        }

        socket.onerror = function (err) {
            console.log(„Ein Fehler ist aufgetreten.“);
        }

        socket.send(„Hallo Welt“);
}
```

LOCAL STORAGE API, SESSION STORAGE API, INDEXED DB API

WEB STORAGES - LOKALES SPEICHERN VON DATEN

- localStorage - Datensicherung ohne Zeitlimit
- sessionStorage - Datensicherung für eine Session
- In Folgenden werden die Methoden anhand des localStorage erläutert, der sessionStorage funktioniert analog.

- Bisher wurde das mit Cookies umgesetzt. Cookies eigenen sich aber nicht für das Handling von größeren Datenmengen, da sie bei JEDEM Request an den Server abgerufen werden. Das ist langsam und wenig effizient.

- HTML5 ruft die Daten nicht bei jedem Aufruf ab, sondern nur bei einer Abfrage. So stellt auch das Handling größerer Datenmengen keine Beeinträchtigung der Performance dar.

- Daten werden in einem eigenen, isolierten Bereich pro Domain gespeichert und sind diesem zugeordnet.
- Zum Speichern und Abrufen der gespeicherten Daten wird Javascript verwendet.

EINEN LOCALSTORAGE ERZEUGEN UND ABRUFEN:

```
<script type="text/javascript">
   localStorage.lastname="Smith";
   document.write(localStorage.lastname);
</script>
```

LOCALSTORAGE - FUNKTIONEN

```
// Speichern, lesen und löschen erfolgt über Javascript
// Funktionen und dem localStorage Objekt.

localStorage.setItem (key, value);
localStorage.getItem (key);
localStorage.removeItem (key);
```

EIN LOKALER SEITENZÄHLER

```
if (localStorage.pagecount)  {
   localStorage.pagecount
      =Number(localStorage.pagecount) +1;
} else {
   localStorage.pagecount=1;
}

document.write("Visits "
   + localStorage.pagecount
   + " time(s)."
);
```

ARBEITEN MIT DATEIEN IM BROWSER

FILE API, DATEIEN / DATEILISTEN / BLOBS, FILEREADER, FILEWRITER, DRAG&DROP

DIE FILE API

- Mit der Einführung von HTML5 wurden auch die Fähigkeiten von JavaScript ausgebaut. So gibt es mit der File-API eine Möglichkeit, auf lokale Dateien zuzugreifen und diese im HTML-Dokument darzustellen. Auf diese Weise lassen sich Dateien beispielsweise vor einem Upload im Browserfenster darstellen und überprüfen.

FILE API'S 3 TEILE

- Dateien lesen und bearbeiten: File/Blob, FileList, FileReader
- Erstellen und schreiben: BlobBuilder, FileWriter
- Verzeichnisse und Dateisystemzugriff: DirectoryReader, FileEntry/DirectoryEntry, LocalFileSystem

FILE/BLOB

FILELIST

FILEREADER

BLOBBUILDER

FILEWRITER

DIRECTORYREADER

LOCALFILESYSTEM

EIN DATEISYSTEM ANFRAGEN

- window.requestFileSystem() fordert Zugriff auf ein durch die Sandboxing-Technologie geschütztes Dateisystem an:

```
window.requestFileSystem  = window.requestFileSystem ||
window.webkitRequestFileSystem;
```

- ```
  window.requestFileSystem(
   type,
   size,
   successCallback,
   opt_errorCallback
  );
  ```
  ```

TYPE: PERSISTENT ODER TEMPORARY

- window.TEMPORARY oder window.PERSISTENT.
- Daten unter TEMPORARY können vom Browser bei Bedarf entfernt werden, beispielsweise wenn mehr Speicherplatz benötigt wird.
- PERSISTENT Speicher kann nicht vom Browser gelöscht werden, nur vom Nutzer oder von der App.
- Für PERSISTENT muss vom Nutzer ein Kontingent eingeräumt werden.

SIZE

- Die von der Anwendung benötigte Größe (in Byte) für den Speicher.
- 5 * 1024 * 1024 => 5Mb

SUCCESSCALLBACK NACH DEN ANLEGEN DES SPEICHERS

- successCallback
  Callback, das bei einer erfolgreichen Anforderung eines Dateisystems aufgerufen wird. Das Argument ist ein FileSystem-Objekt.

ERRORCALLBACK IM FEHLERFALL

- Optionales Callback für die Behandlung von Fehlern oder bei der Ablehnung einer Anforderung zum Dateisystemabruf.
- Das Argument ist einFileError-Objekt.

BEISPIEL

- function onInitFs(fs) {
  console.log('Opened file system: ' + fs.name);
  }
  window.requestFileSystem(
  window.TEMPORARY, 5*1024*1024, onInitFs,
  errorHandler
  );

- function errorHandler(e) {
    var msg = '';
    switch (e.code) {
        case FileError.QUOTA_EXCEEDED_ERR:
            msg = 'QUOTA_EXCEEDED_ERR';
            break;
        case FileError.NOT_FOUND_ERR:
            msg = 'NOT_FOUND_ERR';
            break;
        case FileError.SECURITY_ERR;
            msg = 'SECURITY_ERR';
            break;
        case FileError.INVALID_MODIFICATION_ERR:
            msg = 'INVALID_MODIFICATION_ERR';
            break;
        case FileError.INVALID_STATE_ERR:
            msg = 'INVALID_STATE_ERR';
            break;
        default:
            msg = 'Unknown Error';
            break;
    };
    console.log('Error: ' + msg);
}

SPEICHER-KONTINGENT ANFORDERN

- PERSISTENTer Speicher erfordert eine Nutzerberechtigung.
- TEMPORARY Speicher braucht dies nicht, da der Browser die Daten bei Bedarf entfernen kann.
- Unter Chrome gibt es für PERSISTENTen Speicher unter webkitStorageInfo über ein neues API zur Anforderung von Speicher:

```
window.webkitStorageInfo.requestQuota(
  PERSISTENT, 1024*1024,
  function(grantedBytes) {
        window.requestFileSystem(
              PERSISTENT, grantedBytes, onInitFs, errorHandler
        );
  }, function(e) {
  console.log('Error', e);
});
```

- Nach einmaliger Berechtigung muss requestQuota() nicht mehr aufgerufen werden.
- Nachfolgende Aufrufe unterliegen einer NOOP-Anweisung (No Operation Performed), sodass kein Vorgang ausgeführt wird.
- Darüber hinaus gibt es ein API zur Abfrage der aktuellen Kontingentnutzung und -zuweisung des Ursprungs:

```
window
 .webkitStorageInfo
 .queryUsageAndQuota().
```

SPEICHERHANDLING

- requestFileSystem() fordert ein Speicherkontigent an, in der Regel ässt ein Browser 5 Mb pro Domain zu.
- Das Dateisystem ist durch die Sandboxing-Technologie geschützt ist, das heißt, dass eine Web-App nicht auf die Dateien einer anderen App zugreifen kann.
- Es kann keine Dateien lesen bzw. in einen beliebigen Ordner auf der Festplatte des Nutzers – beispielsweise "Eigene Bilder", "Eigene Dokumente" und so weiter – schreiben.

FILEENTRY/DIRECTORYENTRY

DAS FILEENTRY OBJEKT

- Es enthält Methoden und Eigenschaften für den Zugriff auf Dateieinträge:

- fileEntry.isFile === true
  fileEntry.isDirectory === false
  fileEntry.name
  fileEntry.fullPath
  ...
  fileEntry.getMetadata(successCallback, opt_errorCallback);
  fileEntry.remove(successCallback, opt_errorCallback);
  fileEntry.moveTo(dirEntry, opt_newName, opt_successCallback, opt_errorCallback);
  fileEntry.copyTo(dirEntry, opt_newName, opt_successCallback, opt_errorCallback);
  fileEntry.getParent(successCallback, opt_errorCallback);
  fileEntry.toURL(opt_mimeType);
  fileEntry.file(successCallback, opt_errorCallback);
  fileEntry.createWriter(successCallback, opt_errorCallback);
  ...

SO WIRD EINE DATEI I FILESYSTEM ANGELEGT

- Sie können im Dateisystem mithilfe von getFile(), einer Methode der DirectoryEntry-Schnittstelle, eine Datei suchen oder erstellen. Nach der Anforderung dieser Datei wird ein FileSystem-Objekt an das Erfolgs-Callback weitergegeben. Dieses Objekt enthält eineDirectoryEntry (fs.root)-Klasse, die auf das Stammverzeichnis des Dateisystems der App verweist.

- // Erstellen einer leeren Datei "log.txt" im Stammverzeichnis der App:

```
function onInitFs(fs) {
  fs.root.getFile('log.txt', {create: true, exclusive: true}, function(fileEntry) {
        // fileEntry.isFile === true
        // fileEntry.name == 'log.txt'
        // fileEntry.fullPath == '/log.txt'
  }, errorHandler);
}
window.requestFileSystem(
  window.TEMPORARY, 1024*1024, onInitFs, errorHandler
);
```

DATEIEN LESEN

- // Aufruf der Datei "log.txt" aufgerufen, Inhalt mit dem FileReader-API lesen
  // Inhalt in eine Textarea schicken, Fehlermeldung, falls Datei nicht existiert.

```
function onInitFs(fs) {
    fs.root.getFile('log.txt', {}, function(fileEntry) {
            // übernimmt ein File objekt, FileReader liest den Inhalt
            fileEntry.file(function(file) {
                    var reader = new FileReader();
                    reader.onloadend = function(e) {
                            var txtArea = document.createElement('textarea');
                            txtArea.value = this.result;
                            document.body.appendChild(txtArea);
                    };
                    reader.readAsText(file);
            }, errorHandler);
    }, errorHandler);
}
window.requestFileSystem(window.TEMPORARY, 1024*1024, onInitFs, errorHandler);
```

EINE DATEI SCHREIBEN

```
// Leere Datei "log.txt" erstellen, falls sie noch nicht existiert, Mit "Lorem ipsum" füllen.
function onInitFs(fs) {
    fs.root.getFile('log.txt', {create: true}, function(fileEntry) {
            // Ein FileWriter Objekt für den Dateieintrag "log.txt" erstellen.
            fileEntry.createWriter(function(fileWriter) {
                    fileWriter.onwriteend = function(e) {
                            console.log('Write completed.');
                    };
                    fileWriter.onerror = function(e) {
                            console.log('Write failed: ' + e.toString());
                    };
                    // Ein neues Blob erstellen und Inhalte schreiben.
                    var bb = new BlobBuilder(); // Note: window.WebKitBlobBuilder in Chrome 12.
                    bb.append('Lorem Ipsum');
                    fileWriter.write(bb.getBlob('text/plain'));
            }, errorHandler);
    }, errorHandler);
}
window.requestFileSystem(window.TEMPORARY, 1024*1024, onInitFs, errorHandler);
```

DATEN FORTSCHREIBEN

```
// Mit dem folgenden Code wird der Text "Hello World" an das Ende unserer
// Datei angehängt.
// Es wird ein Fehler ausgelöst, falls die Datei nicht vorhanden ist.

function onInitFs(fs) {
    fs.root.getFile('log.txt', {create: false}, function(fileEntry) {
            // Create a FileWriter object for our FileEntry (log.txt).
            fileEntry.createWriter(function(fileWriter) {
                fileWriter.seek(fileWriter.length); // Start write position at EOF.
                // Create a new Blob and write it to log.txt.
                var bb = new BlobBuilder();
                bb.append('Hello World');
                fileWriter.write(bb.getBlob('text/plain'));
            }, errorHandler);
    }, errorHandler);
}
window.requestFileSystem(window.TEMPORARY, 1024*1024, onInitFs,
errorHandler);
```

MEHRERE DATEIEN HOCHLADEN UND INS DATEISYSTEM DES BROWSERS KOPIEREN

- <input type="file" id="myfile" multiple />
```
document.querySelector('#myfile').onchange = function(e) {
    var files = this.files;
    window.requestFileSystem(window.TEMPORARY, 1024*1024, function(fs) {
            // Alle von Nutzer ausgewählten Dateien übertragen.
            for (var i = 0, file; file = files[i]; ++i) {
                    // Capture current iteration's file in local scope for the getFile() callback.
                    (function(f) {
                            fs.root.getFile(file.name, {create: true, exclusive: true},
  function(fileEntry) {
                            fileEntry.createWriter(function(fileWriter) {
                                    fileWriter.write(f); // Note: write() can take a File or Blob object.
                            }, errorHandler);
                            }, errorHandler);
                    })(file);
            }
    }, errorHandler);
};
```

DATEIEN ENTFERNEN

```
// Mit dem folgenden Code wird
// die Datei "log.txt" gelöscht.
window.requestFileSystem(
 window.TEMPORARY, 1024*1024, function(fs) {
    fs.root.getFile('log.txt', {create: false},
        function(fileEntry) {
            fileEntry.remove(function() {
            console.log('File removed.');
        }, errorHandler);
    }, errorHandler);
}, errorHandler);
```

DAS DIRENTRY OBJEKT MIT VERZEICNISSEN ARBEITEN

- // Eigenschaften und Methoden von DirectoryEntry:
  dirEntry.isDirectory === true
  // Vergleiche auch mit dem fileEntry Objekt.
  ...
  var dirReader = dirEntry.createReader();
  dirEntry.getFile(
    path, opt_flags, opt_successCallback, opt_errorCallback
  );
  dirEntry.getDirectory(
    path, opt_flags, opt_successCallback, opt_errorCallback
  );
  dirEntry.removeRecursively(
    successCallback, opt_errorCallback
  );
  ...

EIN VERZEICHNIS ERSTELLEN

- // Mit getDirectory() aus DirectoryEntry werden
  // Verzeichnisse gelesen oder erstellt.
  // Entweder Namen oder Pfad als zu suchendes
  // bzw. zu erstellendes Verzeichnis angeben.
  // Hier: ein Verzeichnis "MyPictures" erstellen:

```
window.requestFileSystem(
 window.TEMPORARY, 1024*1024, function(fs) {
      fs.root.getDirectory('MyPictures', {create: true},
      function(dirEntry) {
          ...
           }, errorHandler);
}, errorHandler);
```

ANLEGEN EINES FILESYSTEMS

```
window.requestFileSystem(window.TEMPORARY, 1024 * 1024,
function(fs) {
  // fs.root is a DirectoryEntry object.
  fs.root.getFile('log.txt', {create: true},
function(fileEntry) {
    fileEntry.createWriter(function(writer) {  // writer is a
FileWriter object.
      writer.onwrite = function(e) { ... };
      writer.onerror = function(e) { ... };
      var bb = new BlobBuilder();
      bb.append('Hello World!');
      writer.write(bb.getBlob('text/plain'));
    }, opt_errorHandler);
  }
}, opt_errorHandler);
```

DATEIUPLOAD MIT DER FILE API

```
<input type="file" id="dateien" name="dateien"
multiple="multiple" />
<ul id="dateiliste"></ul>

document.getElementById("dateien").addEventListener("change",
dateiausgabe, false);
```

JAVASCRIPT-FUNKTION, DIE DIE INFORMATIONEN ZU DEN DATEIEN HERAUSGIBT:

```
function showFiles(event) {
  var files = event.target.files;
  for (var i = 0, file; file = files[i]; i++) {
    var element = document.createElement("LI");
    var info = document.createTextNode(file.name + " (" +
file.type + "), " + file.size + " bytes");
    element.appendChild(info);
    document.getElementById("filelist").appendChild(element);
  }
}
```

HOCHGELADENE DATEIEN DARSTELLEN

- Statt nur Informationen zu den ausgewählten Dateien auszugeben, lassen sich Dateiinhalte auch darstellen. Dazu gibt es das FileReader-Interface, welches die Ausgabe und Darstellung von Dateiinhalten im HTML-Dokument ermöglicht.

```
function showFilesAsImage(event) {
  var files = event.target.files;
  for (var i = 0, file; file = files[i]; i++) {
    var contents = new FileReader();
    contents.onload = (function(file) {
      return function(e) {
        var element = document.createElement("li");
        var info = document.createTextNode(file.name + " (" +
file.type + "), " + file.size + " bytes");
        var image = document.createElement("img");
        image.setAttribute("src", e.target.result);
        element.appendChild(info);
        element.appendChild(document.createElement("br"));
        element.appendChild(image);
        document.getElementById("fileliste").appendChild(element);
      };
    })(file);
    contents.readAsDataURL(file);
  }
}
```

DRAG&DROP

- Mit JavaScript-Bibliotheken wie jQuery ist es zwar möglich, Drag and Drop innerhalb eines Browsers zu realisieren, will man aber den Desktop einbeziehen oder mit verschiedenen Browsern arbeiten, kommt man an HTML5 nicht vorbei.

- Die dafür vorgesehene Drag-and-Drop-API wurde ursprünglich von Microsoft bereits 1999 mit dem IE5 auf den Markt gebracht, später von Safari und Mozilla übernommen (Chrome unterstützt sie ebenfalls) und fand deshalb Eingang in den HTML5-Standard.

- Eine saubere Cross-Browser-Verwendung macht aber, wie sollte es anders sein, ein wenig Mühe.

- Quelle: http://t3n.de/news/

TÜCKEN UND IMPLEMENTIERUNGSSCHWÄCHEN

- Alle Image- und Anker-Elemente sind per Default „draggable“, also für Drag-and-Drop verwendbar;

- Andere Elemente muss man mit dem HTML-Attribut draggable ausstatten;

- Webkit hält sich nicht an den Standard, hier muss der Draggable-Status über CSS gesetzt werden;

- im IE muss man, da er das draggable-Attribut nicht kennt, mit einem Trick nachhelfen und ein -Element, das draggable sein soll, zu einem <a>-Element umbauen;

- Das HTML-Attribut dropzone für die Zielzone des Drags wird von derzeit keinem Browser unterstützt.

EVENTS FÜR DAS DRAG AND DROP

Verschiedene DOM-Events stehen nun zur Verfügung, um mit dem Drag-and-Drop-Vorgang umzugehen,
zum Beispiel:

dragstart („Jetzt geht's los“),
drag („Es passiert!“)
dragged („Das war's, wir sind fertig“).

DATEN SCHICKEN UND EMPFANGEN

- `set Data(), getData()`
- Über setData lassen sich per Drag-and-Drop auch Daten übertragen, denen man einen MIME-Type mitgeben kann.
- Diese Daten können mit getData wieder ausgelesen werden.

EIN ELEMENT DRAGGABLE MACHEN

Img und Anchor Elemente sind von Haus aus draggable.
Andere Elemente erhalten das Attribut "draggable".

```
<div draggable="true">
    Drag me
</div>
```

WEBKIT BENÖTIGT CSS!

```
<div draggable="true">
    Drag me
</div>

[draggable="true"] {
  -khtml-user-drag: element;
  -webkit-user-drag: element;
  -khtml-user-select: none;
  -moz-user-select: none;
  -webkit-user-select: none;
  user-drag: element;
  user-select: none;
}
```

EIN ELEMENT DROPPABLE MACHEN

von Haus aus: Input, Textarea und alle Elemente, deren Inhalte bearbeitbar sind. Für alles andere gibt es ein "dropzone" Attribut.

```
<div dropzone="..."
     ondrop="handleDrop(event, this)">
    Drop here!
</div>
```

DROPPING

```
var dropzone = document.querySelector(".dropzone");
document.querySelector(".dragr").ondragstart = function(e)
{
    var dt = e.dataTransfer;
    dt.setData("text/plain", "I got dropped");
}
dropzone.ondrop = function(e) {
    var dt = e.dataTransfer,
    elem = this;
    elem.textContent = dt.getData("text/plain");
    e.preventDefault();
}
dropzone.ondragenter = function(e){e.preventDefault();}
dropzone.ondragover = function(e){e.preventDefault();}
```

DRAGGING AUF DEN DESKTOP (DERZEIT NUR CHROME)

```
<a href="assets/html5-cheat-sheet.pdf"
    class="button dragout"
    data-downloadurl="
         application/pdf:
         HTML5CheatSheet.pdf:
         http://url-der-datei/dateiname.pdf"
>Zieh mich auf den Desktop</a>
```

WEITERE BEISPIELE UND HINTERGRÜNDE

- http://www.html5rocks.com/de/tutorials
- http://www.diveintohtml5.com
- http://w3.org/TR
- http://whatwg.org

CACHING FOR OFFLINE, BACKGROUND TASKS, PUSH NOTIFICATION

SERVICE WORKER

SERVICE WORKER

Browser

Service Worker

Internet

Remote Server

SERVICE WORKER

- SW bilden einen programmierbaren Netzwerk-Proxy
- SW können nicht direkt auf das DOM zugreifen
- Werden bei Nicht-Nutzung automatisch beendet
- Verwenden Promises
- Verwenden https (außer bei localhost-Zugriffen)

SERVICE WORKER

- Caching, API Aufrufe
- Push Notifications (Push-/Notification- APIs)
- Hintergrund-Synchronisierung von Daten
- Vorausladen von Daten
- PWA (Progressive Web Apps) - Key Feature

SERVICE WORKER LIFECYCLE

register

install

activate

message events

functional events

fetch,
push,
sync

APPLICATION CACHING
MANIFESTE
CACHE, NETWORK, FALLBACK

CACHING UND OFFLINEFÄHIGKEIT

APPLICATION CACHING

- Web Apps, die offline arbeiten, sind im Web immer noch selten anzutreffen.

- Im folgenden wird die Vorgehensweise erläutert, eine Webbapp offlinefähig gemacht werden kann. Dabei geht es nicht um ein "Hey, es läuft auch im Flieger" - Szenario. Die Offline-Fähigkeit einer Webbapp kann deren Ladeverhalten beschleunigen und sie für instabile Netzwerkvoraussetzungen stabiler machen.

- Für eine grundlegende Einführung in die API empfehlen wir zum Beipiel "Dive into HTML5" von Mark Pilgrim, im folgenden setzen wir Kenntnisse der API voraus.

- Eine App offlinefähig zu machen, erfordert zwei Komponenten: die Daten der App (HTML, Bilder, Styles und Javascript) und die Daten des Users.

OFFLINE USER DATA

- Wir beginnen mit dem local data storage, da unsere erste Implementierung noch geradeaus arbeitet und die Daten im Browser speichert. Bisher wäre dazu ein Cookie zum Einsatz gekommen. Für das Speichern im Browser gibt es zwei bereits weit verbreitete Methoden, sowie eine dritte in der Vorbereitung.

- Die erste Lösung wird von allen modernen Browsern unterstützt, damit sind alle einschließlich des Internet Explorer 8 gemeint. Der localStorage - ein einfacher Schlüssel/Werte-Paar Speicher.

- WebSQL Database ist der zweite, eine für Webkitbrowser entwickelte Spezifikation, die auch von Opera unterstützt wird. Sie basiert auf einem sehr dünnen Layer auf der Basis von SQLite und den vollen relationalen Befehlssatz zur Verfügung stellt. Leider hat Firefox angekündigt, diese Technologie niemals unterstützen zu wollen.

- Die dritte Option ist IndexedDB, im Moment noch nicht fertig spezifiziert und für die Zukunft gedacht.

- Wir wählen für unser Beispiel den localStorage.

LOCALSTORAGE

- Der localStorage speichert Werte als Strings. Das macht es einfach, auch komplexe Daten zum Sichern in ein JSON zu wandeln.

- Cube erledight die meisten Seiten- und Templateberechnungen auf Clientseite. Über GET Request werden Daten und Listen vom Server angefordert, als JSON geliefert und "on the fly" im HTML des DOM verbaut.

- Die Responsedaten des Servers werden gecacht (der Zugang ist der volle GET URL samt Parametern). Wenn wir die Daten ein zweites Mal verwenden, können wir sie bereits aus dem Cache abrufen und unmittelbar anzeigen, während der HTTP Request im Hintergrund nach einem Update forscht und ggf. den Bildschirm aktualisiert.

- Bei bestehender Onlineverbindung erscheint so das Laden der Seite als schnell. Falls keine Netzwerkverbindung existiert, wird dieser Request fehlschlagen, dennoch sind zumindest die Daten des letzten Requests zu sehen.

- Das Offlinebearbeiten unterstützen wir in diesem Beispiel nicht, unsere iOS Apps können auch das. Daten können dort auch offline geändert werden, sie werden zwischengespeichert und bei erneuter Onlineverbindungzum Server gesandt. Dabei werden Synchronisationskonflikte notwendigerweise bearbeitet. Aber dort profitieren wir von den Möglichkeiten einer SQL Datenbank.

- Wir behalten die Entwicklungen der IndexedDB und WebSQL im Auge, da eine Offlinebearbeitung wünschenswert wäre und der localStorage nach unserer Meinung dafür nicht ausreicht.

DER APPLICATION CACHE ALS MANIFEST DATEI

- HTML5 definiert einen Application Cache:
- Er basiert auf einem Manifest, das bestimmt, welche Dateien im Offlinemodus zur Verfügung stehen sollen.

STEUERUNG DES CACHES

- Es exisiteren drei Sektionen innerhalb eines Manifestes:

- CACHE: das, was offline gespeichert werden soll
  NETWORK: das, was immer aus dem Netz geladen werden muss
  FALLBACK: das, was für NETWORK Dateien angezeigt wird,
  wenn der Zugriff auf das Netzwerk nicht funktioniert.

- Außerdem wird jede Seite, die auf das Cache Manifest verweist, im Cache als MASTER Eintrag hinzugefügt, was in jedem Fall ein Cachingverhalten auslöst.

BEISPIEL EINES APPLICATION CACHES

- <html manifest="example.appcache">
- Die Testdatei wird im HTML Tag verankert. Der Webserver muss auf die Verwendung der Datei Endungen Manifest eingerichtet werden.
- Der Browser arbeitet die Datei ab und erstellt Requests für alle Einträge.
- window.applicationCache.addEventListener('checking', updateCacheStatus, false);

- Ein Manifest kennt drei Sektionen:
- #version 1
- CACHE:
  # files to cache
  about.html
  html5.css
  index.html
- NETWORK:
  signup.html
- FALLBACK:
  signup.html offline.html
  /app/ajax/ ajax-fallback.html

- Zunächst ist zu sagen dass die {{ }} und {% %} Zeilen keiner Spezifikation entsprechen. Aber wir möchte unser Manifest dynamisch halten und setzen dafür Django's Template Engine für Serverseitige Template Verarbeitung ein.

- Warum benötigen wir ein dynamisches Manifest? Für eine Antwort müssen wir betrachten, wie ein Browser mit dem Netzwerk interagiert, wenn ein Cache Manifest existiert.

- Stellen Sie sich vor, sie kommen auf http://cube.bitrzr.com/, haben sich zuvor eingeloggt und befinden sich jetzt im Offlinemodus. Die Seite befindet sich also im Offline Cache, der Server wird keine neue Version ausliefern.

- Im Hintergrund wird der Browser das Manifest laden und die geladene Version damit Byte per Byte vergleichen. Sogar Kommentare werden dabei berücksichtigt. Wenn alles übereinstimmt, geschieht nichts weiter.

- Falls aber Unterschiede gefunden werden, dann wird das Manifest neu eingelesen und alle Resourcen werden neu geladen - eigentlich startet der normale HPPT Prozess, wenn also die expires header richtig gesetzt sind, dann werden tatsächlich nur aktualisierte Dateien neu geladen.

- Erinnern Sie sich, dass wir sagten die meisten Templateangelegenheiten werden auf clientseitig verarbeitet? Es gibt aber einige Dinge die immer noch vom Server über die HTML Seite geliefert werden, Zum Beispiel die Links am Ende der Seite oder der Logout Link oben links. Und das möchten wir auch so beibehalten.

- Stellen Sie sich vor, Sie hätten ein nicht-dynamisches Cache Manifest, der User stellt über die settings Seite die Sprache von Englisch nach Französisch. Er lässt die Seite neu und der Logout ist immer noch in Englisch, Weil sich ja das Cache Manifest nicht verändert. Und solange es sich nicht ändert wird die Seite wieder wie zuvor geladen.

- Idealerweise sollte der Browser von alleine seinen Cache aktualisieren. Das Cache manifest verhindert dies aber, und es gibt keine Funktion zur Konfiguration. Solange das cache manifest sich nicht verändert, wird der Browser keine neuen, aktualisierten Daten vom Server anfordern.

- Was wir also tun müssen, ist die Einstellungen zu sichern. Dies erzeugt einen post requestp an den Server, dabei ändern wir einen Wert im Profil des Users. Nach dem Ausführen des requests sagen wir den Browser, dass der Cache neu geladen werden muss. Dafür gibt es die Funktion window.applicationCache.update().

- Außerdem ändern wir eine Zeile im Manifest. Wenn wir dies über einen Templatemechanismus machen, geschieht dies automatisch. Andernfalls müssen wir manuell zum Beispiel die Versionsnummer ändern. Damit veranlassen wir den application cache nach Änderungen in den HTML Dateien zu suchen.

- Far Future Expires

- Before we continue looking at our manifest, let's take a second to discuss HTTP resource caching.
  If you tell a browser that a resource won't expire till the end of times the browser won't even check if the resource has changed, and so you save a round trip to the server. Problem is, what if the resource actually changes?

- Well, a way to work around that is to make sure the resource's URL also changes. So, for instance, let's say all your app's code is in a file called all.js; instead of serving it from /media/all.js, you could serve it from /media/version1/all.js. If you made changes to the file, you could drop it into /media/version2/all.js and change all references to that new URL. The browser would actually see a new resource, and whatever caching was specified for version 1 wouldn't be in effect.

- A slightly smarter and less error prone way to do this would be to calculate a checksum of the file and use that instead of the version; the process could then automated, and no mistakes would be possible. So the file would be served from/media/ABCD5443/all.js and if it was changed, the fingerprint would also change, and it would start being served from something completely, like /media/EFAB3431/all.js.

- That's exactly what we're doing. During our build process, after we minify resources et al, we calculate the fingerprint of each of the static resources, and store it in a dictionary. Later on, a custom Django tag, fingerprint, reads the value from that dictionary, and outputs the tagged URL. Back to the cache manifest, where you see:

- /media/{% fingerprint "main.min.css" %}

- The server will actually send the client something like:

- /media/343423ef/main.min.css

- And that's also the URL being referenced from the HTML files. So, even without using the fancy HTML5 app cache, we could reduce the amount of required requests to the server, and speed up page loading.

- Offline Resource Versioning
- Back to the offline app manifest - what happens if you change a JavaScript file that's available offline? Well, nothing - the browser won't check for changes if the manifest doesn't change, right?
- That's why some people recommend you include a comment with a version number on your manifest, something like this:
- # Cube Offline Manifest v34
- The idea is that when you make changes to the resources you increase this number. Well, if you go back to our manifest you'll notice we don't include a version field. And we think you also shouldn't.
- Problem with this approach is that it's fragile. What happens if someone changes a file but forgets to update the version? You could come up with a way to automatically update it, but we suggest a different approach: implement the far future expires method explained above.
- That way, each time you change a resource its fingerprint will change, and that will cause the manifest file to change, triggering the resource checking. And since the static resources will be cacheable till the end of time only the file that actually changed will be redownloaded.
- Fallback URLs
- We're not especially satisfied is with the way we're handling offline fallbacks. You see, what we'd like to have is a way to say that if we are offline and the browser is asking for a JSON resource we'd like to return a specific file (in JSON format), else we'd like to return an HTML file.
- Since that kind of MIME type sniffing is not possible, and since we didn't want to explicitly list all the different JSON or HTML endpoints in the cache manifest (careful: there's an implicit * at the end of each entry, and you can't have conflicts - each URL must be covered by only one rule), we're including a magic meta tag in our offline.html file, and if a JSON call fails with a parse error we check for that value, and act accordingly.
- Conclusion
- We're quite happy with our current offline implementation; it gives our users significant functionality and performance advantages.

EIGENSCHAFTEN DES APPLICATIONCACHE OBJEKTES

- window.applicationCache.status
  UNCACHED: 0
  IDLE: 1
  CHECKING: 2
  DOWNLOADING: 3
  UPDATEREADY: 4
  OBSOLETE: 5

METHODEN DES APPLICATIONCACHE OBJEKTES

- update:
  function update() { }
- swapCache:
  function swapCache() { }
- abort:
  function abort() { }
- addEventListener:
  function addEventListener() { }
- removeEventListener:
  function removeEventListener() { }
- dispatchEvent:
  function dispatchEvent() { }

FORMATE, STEUERUNG, MÖGLICHKEITEN

AUDIO UND VIDEO MIT HTML5

DAS VIDEO - ELEMENT

- Oft wird das Video-Tag in den Medien als eine Flashalternative dargestellt, aber es ist mehr als das.

- Sein Hauptvorteil liegt in der natürlichen Integration in die anderen Ebenen der Web-Entwicklung, wie CSS und JavaScript und die anderen HTML-Elemente.

- Im Folgenden wird das Video-Tag und verschiedene Beispiele für unterschiedliche Integrationen in andere HTML5-Funktionen, beispielsweise <canvas> gezeigt.

-

DIE AUSZEICHNUNG <VIDEO>

- `<video src="movie.webm"></video>`

- Dies ist die einfachste Form, eine Videodatei in ein HML Dokument einzubinden. Die Syntax ähnelt dem Image_Element.

- Dies wird in naher Zukunft, wenn alle Browser ein gemeinsames Videoformat unterstützen, die meistgenutzte Syntax sein.

-

<VIDEO> UND <SOURCE>

```
<video>
  <source src="movie.mp4"
   type='video/mp4;   codecs="avc1.42E01E, mp4a.40.2"'
/>
  <source src="movie.webm"
   type='video/webm;
   codecs="vp8, vorbis"'
/>
</video>
```

In diesem Snippet wird das <source>-Tag verwendet, mit dem Sie mehrere Formate als Fallback-Typen aufnehmen können für den Fall, dass der Browser des Nutzers eines davon nicht unterstützt.

EINSTELLEN DES SERVERS FÜR VIDEOS

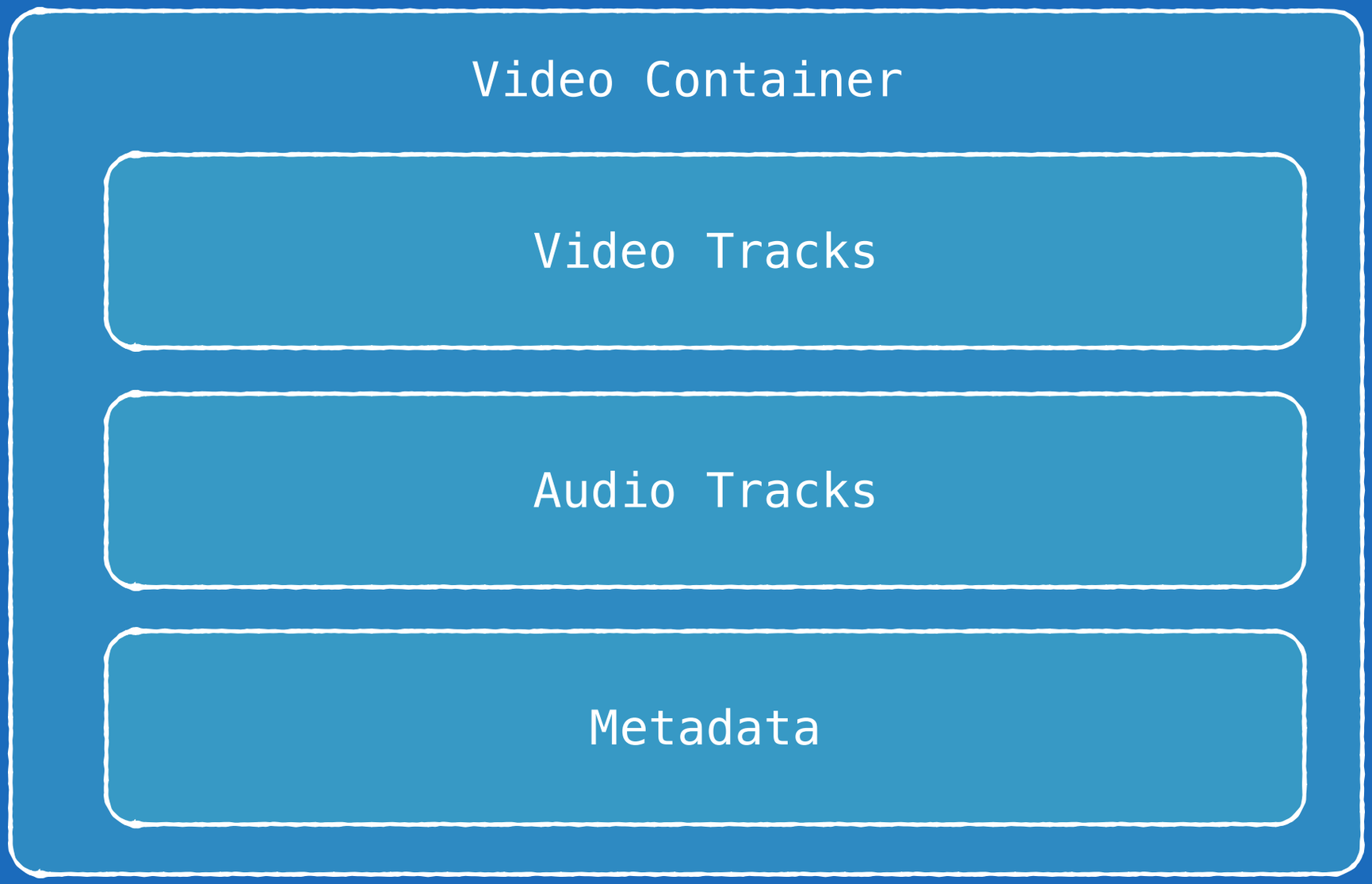
- Der Webserver muss Videodateien mit dem korrekten MIME-Typ im Content-Type-Header bereitstellen. Im anderen Fall funktionieren die Videos möglicherweise nicht ordnungsgemäß.

- In einer Apache-Datei "httpd.conf" werden beispielsweise diese Zeilen hinzugefügt:

- `AddType video/ogg .ogv`
- `AddType video/mp4 .mp4`
- `AddType video/webm .webm`
-

VIDEOCONTAINER

- Eine Audio- oder Videodatei ist eine Containerdatei, ähnlich einem Zip-Archiv, das mehrere Dateien enthält.



VIDEOCONTAINER

- Die Audio und Videospuren werden zur Laufzeit zusammengefügt.

- Die Metadaten enthalten Informationen über den Film, wie das Cover, den Titel, Untertitel und so weiter.

VIDEOFORMATE

- Die drei Formate, die für das Web gebraucht werden, sind "webm", "mp4" und "ogv":

- `.mp4` = H.264 + AAC
- `.ogg/.ogv` = Theora + Vorbis
- `.webm` = VP8 + Vorbis

- Diese sind als Standardformate definiert.
  Dies heisst aber nicht, dass Browser nicht auch andere Formate abspielen können.

AUDIO- UND VIDEOCODECS

- Codecs (coders/decoders) sind Algorithmen, mit denen Audio- und Videodateien zum Abspielen codiert bzw. decodiert werden.

- Sie komprimieren die Daten, so dass die zum Beispiel über das Netz übertragen werden können

- Ohne passenden Decoder kann der Empfänger die Datei nicht öffnen.

BROWSERUNTERSTÜTZUNG

- Heute unterstützen alle modernen Browser HTML5-Video, auch IE9.
- Für Firefox, Chrome und Opera wurde in Bezug auf Codecs vereinbart, über das WebM-Projekt zusätzlich ".webm" als gemeinsames Videoformat zu unterstützen.
- Internet Explorer unterstützt dies ebenfalls, sofern der Codec auf dem Computer installiert wird.

EINSCHRÄNKUNGEN

- Kein Streaming Audio oder Video, da im Moment kein Standard für Bitraten existiert. In Zukunft kann sich das ändern
- Es gelten die Regeln des Cross Site Managements.
- Vollbildvideo ist nicht programmierbar, da es gegen Sandbox-Sicherheitsrichtlinien verstößt. Über den Browser kann es aber eingestellt werden.
- Barrierefreiheit für Audio- und Videodaten ist noch nicht vollständig spezifiziert. Die Spezifikation "WebSRT" für Untertitelunterstützung auf Basis des SRT Formates ist in Arbeit.

PRÜFEN, OB <VIDEO> ZUR VERFÜGUNG STEHT

```
var hasVideo = (document.createElement('video').canPlayType);
```

Erzeugt dynamisch ein Video Element und prüft, ob die Methode canPlay vorhanden ist. So kann bei fehlender Unterstützung eine Fallbackposition, zum Beipsiel für ein Flashvideo eingebaut werden.

ALTERNATIVE AUSGABE

```
<video src="video.ogg" controls>
   Your browser does not support HTML5 video.
</video>
```

VIDEO FÜR ALLE BROWSER HEUTE

```
<video>
  <source src="movie.webm" type='video/webm; codecs="vp8, vorbis"' />
  <source src="movie.mp4" type='video/mp4; codecs="avc1.42E01E, mp4a.
40.2"' />
  <source src="movie.ogv" type='video/ogg; codecs="theora, vorbis"' />
  Video tag not supported.
  Download the video <a href="movie.webm">here</a>.
<video>

Hinweis: Aufgrund eines Fehlers im iPad
müssen Sie ".mp4" als erste Option angeben,
damit das Video auf diesem Gerät geladen wird.
Dies gilt, bis der Fehler behoben ist.
```

STANDARDVIDEO

Da sich fast alle Browseranbieter darauf geeinigt haben, ein gemeinsames Videoformat zu unterstützen, wird dieser Code im Web verwendet werden:

```
<video>
   <source   src="movie.webm"
             type='video/webm;
                   codecs="vp8, vorbis"' />
   <source   src="movie.mp4"
             type='video/mp4;
                   codecs="avc1.42E01E, mp4a.40.2"' />

   Video tag not supported.
   Download the video <a href="movie.webm">here</a>.
<video>
```

MP4 IST PROBLEMATISCH

- Zu den größten Bedenken bei der Verwendung des MP4-Formats zählt die Tatsache, dass sein Video-Codec (h.264) kein offener Codec ist und ein Unternehmen für seine Nutzung Lizenzen bezahlen müsste.

- Ein weiteres Problem mit dem MP4-Format besteht darin, dass das type-Attribut je nach dem für die Codierung des Videos verwendeten Profil spezifischer sein muss als bei anderen Formaten.

- Obwohl es in aller Regel "avc1.42E01E, mp4a.40.2" lautet, sollten Sie nochmals in dieser Profilliste nachsehen, um sich zu vergewissern.

-

DAS VIDEO ELEMENT IN ALTEN IE VERSIONEN?

- Das Google Chrome Frame Plugin

- versetzt Internet Explorer nach seiner Installation in die Lage, die neuesten HTML-, JavaScript- und CSS-Standardfunktionen zu unterstützen.

- Für Entwickler bietet dieses Plug-in den Zusatznutzen, dass sie damit Anwendungen mit modernen Webfunktionen codieren können, auf die auch die IE-Nutzer nicht verzichten müssen. Stellen Sie sich nur vor, wie viel Zeit ein Webentwickler einspart, wenn er keine Hacks und Behelfslösungen für IE codieren muss.

FLASH-FALLBACK

- Sie können auch Flash als Fallback verwenden. Je nach dem Format Ihres Videos müssen Sie dieses unter Umständen erneut codieren, damit es ein mit Flash Player kompatibles Format besitzt.

- Die gute Nachricht ist, dass Adobe sich dazu verpflichtet hat, das webm-Format in Flash Player zu unterstützen.

-
```
<video src="video.ogg">
 <object data="videoplayer.swf"
         type="application/x-shockwave-flash">
   <param name="movie" value="video.swf"/>
 </object>
</video>
```

<VIDEO> UND HTML

VIDEO UND ANDERES HTML

```
<video  poster="star.png"
       autoplay loop controls tabindex="0">
    <source src="movie.webm"
    type='video/webm; codecs="vp8, vorbis"' />
    <source src="movie.ogv"
    type='video/ogg; codecs="theora, vorbis"' />
</video>

Hier wird ein  tabindex verwendet, damit über
die Tastatur auf den Player zugegriffen werden kann.
```

ATTRIBUTE DES VIDEO - ELEMENTES

- `autoplay`
  für die automatische Wiedergabe eines Video- oder Audio - Elementes.

- `loop`
  Stellt die Mediendatei auf
  automatische Wiederholung ein.

- `currentTime`
  Gibt die aktuelle Zeit in Sekunden zurück, die seit dem Start des Films vergangen ist. CurrentTime kann auch gesetzt werden, um eine bestimmte Position im Film anzusteuern.

- `preload`
  Das Video wird im Hintergrund heruntergeladen, sobald die Seite geladen wird, selbst wenn seine Wiedergabe noch nicht begonnen hat.

-

WEITERE ATTRIBUTE

- `poster`
  Mit dem poster-Attribut wird angegeben, welches Bild angezeigt wird, wenn das Video am Anfang geladen wird.

- `controls`
  Der Browser soll automatisch Steuerelemente rendern. (Es werden keine benutzerspezifischen Steuerelemente erstellt.)

- `width, height`
  Read or set the visual display size. This may cause centering, letterboxing, or pillaring if the set width does not match the size of the video itself.

- `videoWidth, videoHeight`
  Return the intrinsic or natural width and height of the video. They cannot be set.

PROGRAMMIERBARE EIGENSCHAFTEN

- `volume`
  Stellt die Lautstärke des Films auf einen Wert zwischen 0.0 und 1.0. Der wert kann auch abgefragt werden.

- `muted`
  Stellt den Film auf stumm oder hörbar und liefert den aktuellen Status von mute zurück.

READ ONLY

- `duration`
  Liefert die volle Läge eines Clips in Sekunden. Fall der Wert unbekannt sein sollte, wird NaN ausgegeben.

- `paused`
  Liefert true, falls der Clip gerade pausiert. Ist per Default auf true gesetzt, wenn der Film noch nicht gestartet wurde.

- `ended`
  Liefert ein true, wenn der Film zu Ende gespielt hat.

- `startTime`
  Liefert die fühestmögliche Startzeit des Films. Das ist normalerweise ein 0.0 Wert, solange der Clip geladen wird und kein alter Content im Buffer liegt.

READ ONLY

- `error`
  Gbit einen Fehlercode aus, falls ein Fehler auftritt.

- `currentSrc`
  Gibt einen String zurück, der den Dateinamen enthält.

<VIDEO> UND JAVASCRIPT

VIDEO UND JAVASCRIPT

- `load()`
  Lädt eine Mediendatei und bereitet sie zum Abspielen vor. Der Befehl wird bei dynamischen geladenen Filmen angewandt. (Nicht bei im HTML stehenden).

- `play()`
  Lädt und spielt einen Film ab. Der Film wird entweder von der Startposition oder der aktuellen Pausenposition abgespielt.

- `pause()`
  Pausiert einen aktiven Film.

- `canPlayType(type)`
  Prüft, ob ein Browser über eine Playmethode verfügt oder nict. Damit kann die Unterstützung des Video - Elementes geprüft werden.

EVENTLISTENER FÜR DAS <VIDEO> ELEMENT

- Wie bei jedem anderen HTML-Element können Sie gängige Ereignisse an das Video-Tag anhängen, beispielsweise Ziehereignisse, Mausereignisse, Fokusereignisse und so weiter.

- Das Videoelement verfügt jedoch über eine Reihe von neuen Ereignissen, anhand derer erkannt und gesteuert wird, wann das Video abgespielt, angehalten oder beendet wird.

```
Events auf Netzwerkebene
loadstart
progress
suspend
abort
error
emptied
stalled

Events auf Zwischenspeicherungsebene
loadedmetadata
loadeddata
waiting
playing
canplay
canplaythrough
```

DER EVENTLISTENER CANPLAY

Auf einfachster Ebene können Sie als Einstieg für Ihre Beschäftigung mit dem Video das "canplay"-Ereignis anhängen.

```
video.addEventListener('canplay', function(e) {
    this.volume = 0.4;
    this.currentTime = 10;
    this.play();
}, false);
```

<VIDEO> UND CSS

VIDEO UND CSS

- Das <video> kann als Toplevel Element des DOM mit allen CSS Attributen ausgestattet werden.

- Dazu gehören Kanten, Deckkraft, aber auch CSS3-Eigenschaften wie Reflexionen, Masken, Farbverläufe, Transformationen, Übergänge und Animationen.

- ```
  #video-css.enhanced {
   border: 1px solid white;
   box-shadow: 0px 0px 4px #ffffff;
   transform: translate(0, 10px);
  }
  ```
  ```

<VIDEO> UND <CANVAS>

VIDEO UND CANVAS

- Canvas ist ein weiteres HTML5-Element, das bei der gemeinsamen Nutzung mit dem Video-Tag eine Fülle von Möglichkeiten bietet.

- Das folgende Beispiel macht sich zwei Funktionen des <canvas>-Elements zunutze, um Bilder zu im- und exportieren. Die erste ist die drawImage-Methode, mit der Bilder aus drei unterschiedlichen Quellen importieren können: aus einem Bildelement, einem weiteren Canvas-Element und einem <video>-Element!

- Bei jeder Ausführung der drawImage() - Methode wird der aktuelle Frame im Video importiert und im Canvas gerendert.

- Die zweite Funktion des verwendeten <canvas>-Tags ist die toDataURL-Methode, mit der Sie den Inhalt des Canvas-Elements in ein Bild exportieren können. Alle anderthalb Sekunden wird ein Bild aus dem Video erstellt. (Das zum Import bzw. Export verwendete Canvas-Element ist verborgen.)

-

- video_dom.addEventListener('play', function() {

```
video_dom.width = canvas_draw.width = video_dom.offsetWidth;
video_dom.height = canvas_draw.height = video_dom.offsetHeight;

var ctx_draw = canvas_draw.getContext('2d');

draw_interval = setInterval(function() {
```

- // import the image from the video

```
    ctx_draw.drawImage(video_dom, 0, 0, video_dom.width, video_dom.height);

    // export the image from the canvas
    var img = new Image();
    img.src = canvas_draw.toDataURL('image/png');
    img.width = 40;
    frames.appendChild(img);

  }, 1500)

}, false);
```

-

EIN VIDEO MIT TRANSFORMATIONEN

Das folgende Beispiel spielt ein Video parallel in einem Canvas Element ab. Dazu wird alle 33 Millisekunden ein Bild aus dem Film in das erste Canvas exportiert.
Ein zweites Canvas Element spielt das Bild erneut ab und legt einige Transformationen darüber. (Dies geht schneller, als die direkte Transformation eines aus dem Film kopierten Bildes.)

```
<!DOCTYPE html>
<html>
  <head>
    <style>
      video, canvas {
        margin-top: 25px;
        width: 200px;
        height: 112px;
        border: 1px solid black;
      }
    </style>
  </head>
  <body>
    <video poster="http://www.html5rocks.com/en/tutorials/video/basics/star.png"
          id="video-canvas-fancy" controls>
      <source src="http://www.html5rocks.com/en/tutorials/video/basics/Chrome_ImF.mp4"
            type='video/mp4; codecs="avc1.42E01E, mp4a.40.2"' />
      <source src="http://www.html5rocks.com/en/tutorials/video/basics/Chrome_ImF.webm"
            type='video/webm; codecs="vp8, vorbis"' />
      <source src="http://www.html5rocks.com/en/tutorials/video/basics/Chrome_ImF.ogv"
            type='video/ogg; codecs="theora, vorbis"' />
    </video>
    <canvas id="canvas-copy-fancy"></canvas>
    <canvas id="canvas-draw-fancy"></canvas>
  </body>
</html>
```

```
video, canvas {
  margin-top: 25px;
  width: 200px;
  height: 112px;
  border: 1px solid black;
}
```

```
    <video poster="http://www.html5rocks.com/en/tutorials/
video/basics/star.png" id="video-canvas-fancy" controls>
      <source src="http://www.html5rocks.com/en/tutorials/
video/basics/Chrome_ImF.mp4" type='video/mp4;
codecs="avc1.42E01E, mp4a.40.2"' />
      <source src="http://www.html5rocks.com/en/tutorials/
video/basics/Chrome_ImF.webm" type='video/webm; codecs="vp8,
vorbis"' />
      <source src="http://www.html5rocks.com/en/tutorials/
video/basics/Chrome_ImF.ogv" type='video/ogg; codecs="theora,
vorbis"' />
    </video>
    <canvas id="canvas-copy-fancy"></canvas>
    <canvas id="canvas-draw-fancy"></canvas>
    <script>
```

```
   var video_dom = document.querySelector('#video-canvas-
fancy');
      var canvas_copy = document.querySelector('#canvas-copy-
fancy');
      var canvas_draw = document.querySelector('#canvas-draw-
fancy');
      var draw_interval = null;
      var ctx_copy = null;
      var ctx_draw = null;

      var offsets = [];
      var inertias = [];
      var slices = 4;
      var out_padding = 100;
      var interval = null;

      var inertia = -2.0;
```

```
video_dom.addEventListener('canplay', function() {
  canvas_copy.width = canvas_draw.width = video_dom.videoWidth;
  canvas_copy.height = video_dom.videoHeight;
  canvas_draw.height = video_dom.videoHeight + out_padding;
  ctx_copy = canvas_copy.getContext('2d');
  ctx_draw = canvas_draw.getContext('2d');
}, false);


    for (var i = 0; i < slices; i++) {
      offsets[i] = 0;
      inertias[i] = inertia;
      inertia += 0.4;
    }

    video_dom.addEventListener('play', function() {
      processEffectFrame();
      if (interval == null) {
        interval = window.setInterval(function()
{ processEffectFrame() }, 33);
      }
```

```
function processEffectFrame() {
  var slice_width = video_dom.videoWidth / slices;
  ctx_copy.drawImage(video_dom, 0 ,0);
  ctx_draw.clearRect(0, 0,  canvas_draw.width, canvas_draw.height);
  for (var i = 0; i < slices; i++) {
    var sx = i * slice_width;
    var sy = 0;
    var sw = slice_width;
    var sh = video_dom.videoHeight;
    var dx = sx;
    var dy = offsets[i] + sy + out_padding;
    var dw = sw;
    var dh = sh;
    ctx_draw.drawImage(canvas_copy, sx, sy, sw, sh, dx, dy, dw, dh);
    if (Math.abs(offsets[i] + inertias[i]) < out_padding) {
      offsets[i] += inertias[i];
    } else {
      inertias[i] = -inertias[i];
    }
  }
};
```

```
video_dom.addEventListener('pause', function() {
      window.clearInterval(interval);
      interval = null;
    }, false);

    video_dom.addEventListener('ended', function() {
      clearInterval(interval);
    }, false);
}, false);
```

<VIDEO> UND <SVG>

<VIDEO> UND <SVG>

- SVG (Scalable Vector Graphic) bietet eine programmatische Möglichkeit zum Rendern und Bearbeiten von Vektorgrafiken, verfügt aber auch über weitere Funktionen wie SVG-Filtereffekte. Mit diesen Filtern können Sie ein spezifisches DOM-Element als Ziel festlegen und einige tolle Effekte wie Weichzeichnen, Mischen, Kacheln und so weiter anwenden.

```
<svg id='image' version="1.1" xmlns="http://www.w3.org/2000/svg">
  <defs>
    <filter id="myblur">
      <feGaussianBlur stdDeviation="1" />
    </filter>
  </defs>
</svg>
<style>
  video { filter:url(#myblur); border: 2px solid red; }
</style>

```

GEOLOCATION API

GEOLOCATION API

- Mit Hilfe der Geolocation API kann der User den seinen geografischen Standort ermitteln lassen. (Um ihn zum Beispiel anderen Usern mitzuteilen).

- Und, wenn er einverstanden ist, kann die Webanwendung ihm dann einen Tipp geben, wo es in der Gegend ein günstiges Angebot für Schuhe gibt.

- Eine andere Idee wäre eine turn-by-turn GPS-style Navigation, oder eine social networking application, die anzeigt, wo sich die Freunde gerade aufhalten.

LÄNGEN- UND BREITENGRADE

- Die Ortsinformation besteht im wesentlichen in der Angabe von Längen- und Breitengrad der aktuellen Position, wie das folgende Beispiel zeigt. Es sind die geographischen Angabe für die Stadt Stuttgart:

- `geographische Breite: 48,755`
  `geographische Länge: –9.175`

- Der nördliche Abstand von Äquator beträgt also 48,755 Grad, die östliche Länge 9,175 Grad ausgehend von Greenwich Meridian, der den Nullmeridian liefert.

- Normalerweise werden die geographischen Angabe in Grad, Minuten und Sekunden angegeben:

- -9° 10' 30" bzw. 48° 47' 30"

WEITERE ANGABEN

- `accuracy, altitude, altitudeAccuracy, heading, speed.`

- Zusätzlich ermittelt die Geolocation API die Genauigkeit der ermittelten Angaben. Ausserdem können je nach Gerät die Höhenangabe, die Genauigkeit der Höhenangabe, die Richtung und die Geschwindigkeit ermittelt werden.

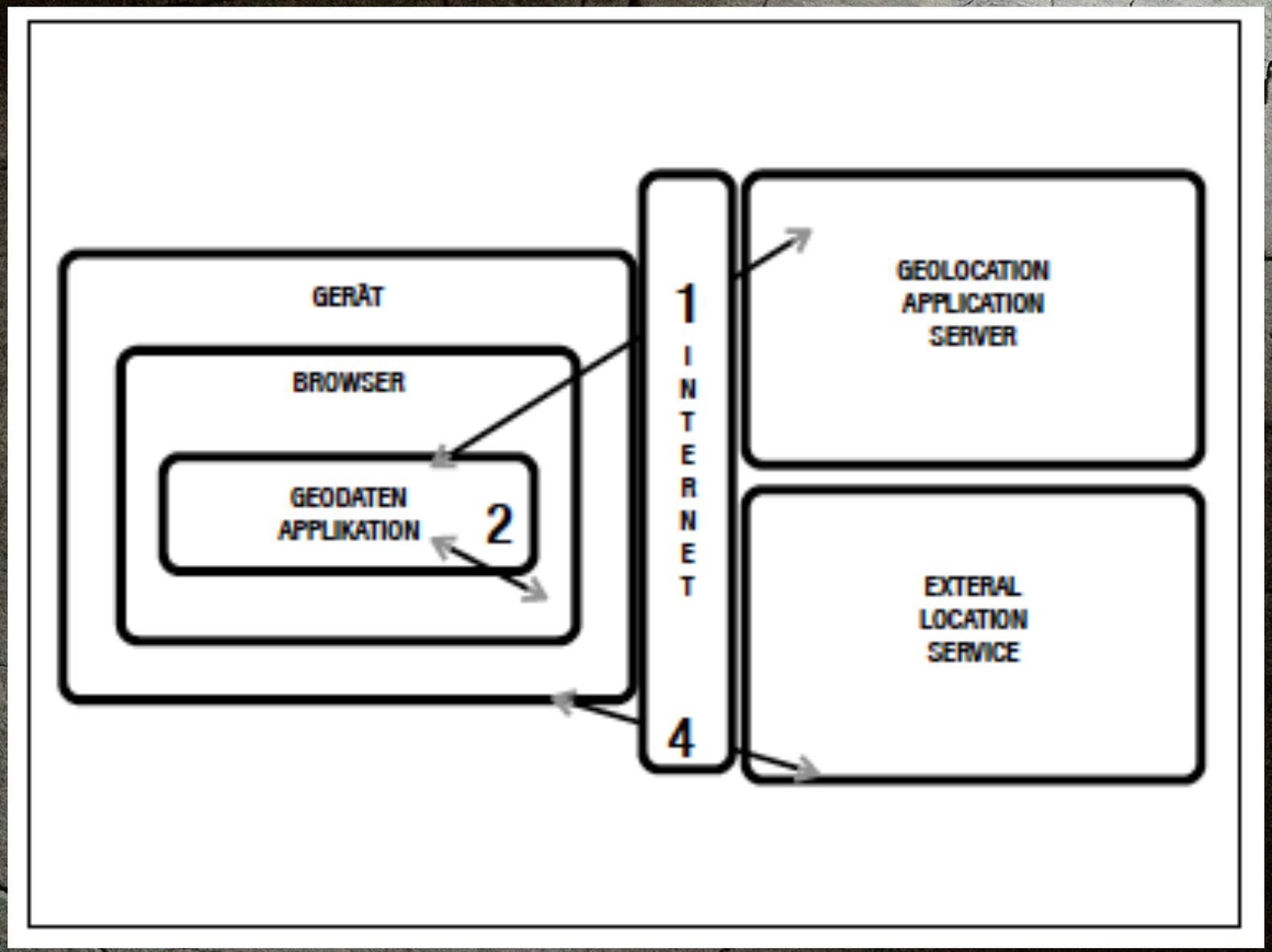
- Wenn keine Werte vorliegen wird jeweils Null zurückgegeben.

- Ein Gerät kann auf folgende Resourcen zur Koordinatenermittlung zurückgreifen:
- IP Adresse
- Koordinatentriangulation (Mittelwertberechnung)
- Global Positioning System (GPS)
- Wi-Fi mit MAC Adressen aus RFID-, Wi-Fi oder
- Bluetoothquellen
- GSM oder CDMA Mobliephone IDs
- Usereingaben
- Viele Geräte verwenden eine Kombination mehrerer Quellen, um eine höhere Genauigkeit im Ergebnis zu erzielen.

SCHUTZ DER PRIVATSPHÄRE

- Die HTML5 Geolocation Spezifikation fordert einen Mechanismus zum Schutz der Privatspäre des Users. So muss der User den Zugriff des Browsers auf seine Geodaten erlauben.

- Allerdings ist offen, was nach der Genehmigung mit den Daten geschieht, oder u.U. auch, wie lange sie gilt.


GERÄT
BROWSER
GEODATEN
APPLIKATION
2
1
INTERNET
4
GEOLOCATION
APPLICATION
SERVER
EXTERAL
LOCATION
SERVICE

- Oft ist es ausreichend, die Position nur einmal abzufragen.
- `navigator.geolocation.getCurrentPosition(updateLocation, handleLocationError);`
- Man kann sie auch permanent aktualisieren
- `navigator.geolocation.watchPosition(updateLocation, handleLocationError);`

GEOLOCATION PRÜFEN

```
function loadDemo() {
   if(navigator.geolocation) {
      document.getElementById("support").innerHTML
      = "HTML5 Geolocation supported.";
   } else {
      document.getElementById("support").innerHTML
      = "HTML5 Geolocation is not supported.";
   }
}
```

FEHLERBEHANDLUNG

```
function handleLocationError(error) {
    switch(error.code){
        case 0: // UNKNOWN ERROR
            updateStatus("There was an error while
            retrieving your location: " +
            error.message);
            break;
    case 1: // PERMISSION DENIED
            updateStatus("The user prevented this
            page from retrieving a location.");
            break;
```

```
        case 2: // POSITION_UNAVAILABLE
        updateStatus("The browser was unable to
        determine your location: " +
        error.message);
              break;
    case 3: // TIMEOUT
              updateStatus("The browser timed out
              before retrieving the location.");
              break;
        }
    }
```