zenika
ARCHITECTURE INFORMATIQUE

HTML5

WebSocket

Plan

- Le protocole
- L'API
- Implémentations coté client et serveur
- Application
- Sécurité

Le protocole websocket

- Spécification normalisée par l'IETF (Internet Engineering Task Force) dans le RFC 6455
- Principe :
  - Communication bi-directionnelle permanente entre un client et un serveur web
    - *Développement d'applications temps-réel*
- Le protocole est constitué de deux parties
  - Handshake
  - Transfert de données

Le protocole websocket

- Handshake
  - Requête client

```
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
```

  - Réponse serveur

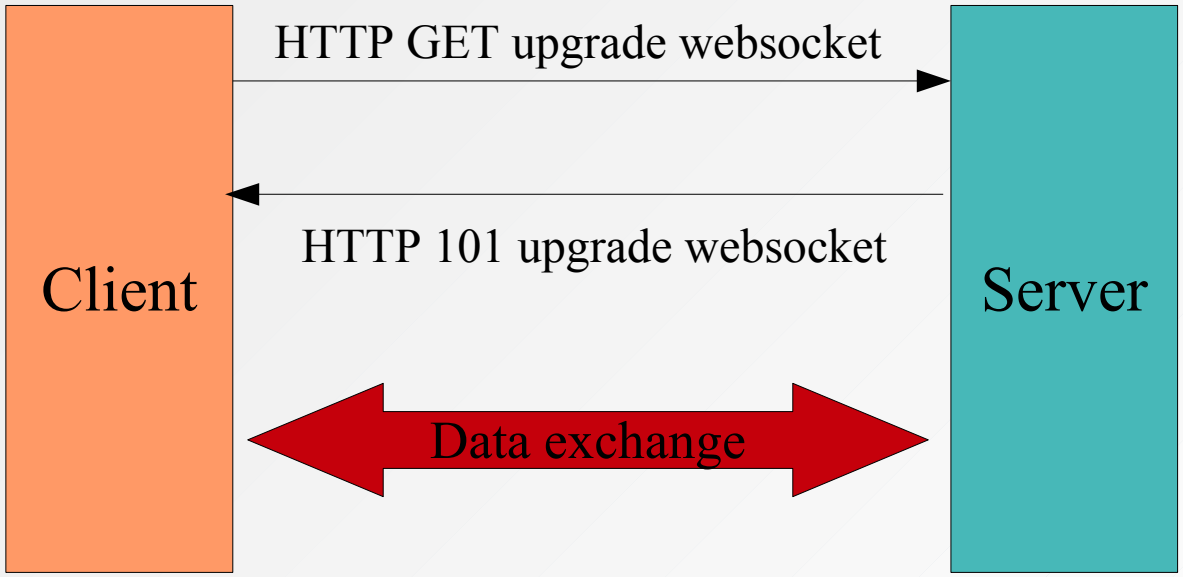
```
HTTP/1.1 101 Switching Protocols
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Accept: s3pPLMBiTxaQ9kYGzzhZRbK+xOo=
Sec-WebSocket-Protocol: chatsuperchat
```

Le protocole websocket

- Suite au handshake, le protocole websocket est établi et l'échange de données peut commencer
- Les données sont envoyés sous forme de « frame » avec un « mask »
  - 4 octets choisis aléatoirement par le client forment le mask
  - Les données sont ensuite « masquées »
  - La trame envoyées sera sous la forme : mask|données
  - Le serveur doit ensuite faire une opération XOR entre les données et le mask

    → assure la sécurité des échange

Le protocole websocket



L'API

- L'API websocket est en cours de spécification par le W3C (draft)

```
[Constructor(in DOMString url, in optional DOMString protocol)]
interface WebSocket {
  readonly attribute DOMString URL;

  // ready state
  const unsigned short CONNECTING = 0;
  const unsigned short OPEN = 1;
  const unsigned short CLOSED = 2;
  readonly attribute unsigned short readyState;
  readonly attribute unsigned long bufferedAmount;

  // networking
           attribute Function onopen;
           attribute Function onmessage;
           attribute Function onclose;
  boolean send(in DOMString data);
  void close();
};
WebSocket implements EventTarget;
```

L'API

- Création d'une instance de websocket

```
var ws = new WebSocket("ws://websocketServerUrl");
```

  - Un second argument optionnel définit des sous-protocoles

- Gestion des événements

```
ws.onopen = function(evt) { alert("Connection open ..."); };
ws.onmessage = function(evt) {
    alert( "Received Message: " + evt.data);
};
ws.onclose = function(evt) { alert("Connection closed."); };
ws.onerror = function(evt) { alert("WebSocket error : " + e.data) };
```

- Envoi de données

```
ws.send(message);
```

- Fermeture de la connexion

```
ws.close();
```

L'API

- Etat de la connexion
  - Attribut `readyState`
    - *0 : connecting → connexion en cours*
    - *1 : open → connexion établie*
    - *2 : closed → fermé ou n'a pu être établie*
- Connexion sécurisée avec wss
  - Chiffrement TLS/SSL similaire à HTTPS

Implémentations Clients

- Navigateurs :
  - Firefox 6.0+
  - Chrome 14.0+
  - Safari 6.0+
  - IE 10
  - Opera 12.1
- Tester le support :

```
if (!!window.WebSocket) {
//open websocket ...
} else {
    console.log("Votre navigateur ne supporte pas les
        websockets");
}
```

- Autres clients et librairies :
  - Socket.io, websocket-node, jetty, jwebsocket, Kaazing, ...

Implémentations Serveurs

- Serveur
  - Java: Jetty, tomcat 7, jwebsocket, GnuWebsocket4J
  - Javascript: websocket node, socketIO (+nodejs)
  - Python : pywebsocket
  - PHP : phpwebsocket
  - C++ : QtWebsocket
  - Kaazing

Application
Chat avec nodejs et websocket

- Serveur

```
var WebSocketServer = require('websocket').server;
var http = require('http');

var clients = [ ]; //client list

var server = http.createServer(function(request, response) {
    //serveur http
});
server.listen(8082, function() {
    console.log((new Date()) + " Server is listening on port
1337");
});

// create the server
wsServer = new WebSocketServer({
    httpServer: server,
});
```

Application
Chat avec nodejs et websocket

- Serveur (suite)

```
wsServer.on('request', function(request) {
    //verifier l'origine puis accepter la connexion
    var connection = request.accept(null, request.origin);
    //subprotocole, origin; retourne une instance de WebSocketConnexion

    var client = clients.push(connection) - 1;

    connection.on('message', function(message) {
        for (var i=0; i < clients.length; i++) {
            clients[i].send(message.utf8Data);
        }
    });

    connection.on('close', function(connection) {
        // close user connection
        console.log((new Date()) + ' Connection closed ');
        clients.splice(client, 1); //remove client
    });
});
```

Application
Chat avec nodejs et websocket

- Client : se connecte à ws://localhost:8082/

```
var ws ;
function loadWebSocket() { //a appeler depuis la page html
    if (window.WebSocket) {
        ws = new WebSocket("ws://localhost:8082/");

        ws.onopen = function() {//alert new user};
        ws.onmessage = function(e) {
            //displayMessage(e.data);
        };
        ws.onclose = function() {//alert close};
        ws.onerror = function(e) {//alert error};

    } else {
        alert("le navigateur ne supporte pas les websockets")
    }
}

function sendMsg(data){ //fonction à appeler pour envoyer un message
    if (ws && ws.readyState == 1) {
        ws.send(data);
    } else {
        alert("websocket fermée");
    }
}
```

Sécurité et fallback

- Sécurité basée sur les origines
  - Connexion du serveur à une origine particulière uniquement
- Données envoyées « masquées » avec la clé
  - Empêchent les attaques d'intermédiaires
- Prévoir un fallback
  - Vers du polling/long-polling si la connexion ne peut être établie

TP