

# HTML5 - WebSockets

Ingo Köster

Diplom-Informatiker (FH)

# Einführung

---

- › Für manche Anwendungen ist es sinnvoll/notwendig, wenn der Server seine Clients aktiv mit Daten versorgen kann
  - › Teilbereiche von Social Media
  - › Business-Lösungen für Banken etc.
  - › Anwendungen im Bereich Sicherheit und Überwachung
  - › (Multiplayer-)Spiele
- › Klassische Web-Anwendungen eignen sich dafür nicht, da HTTP zustandslos ist

# Polling-Verfahren vs. Sockets

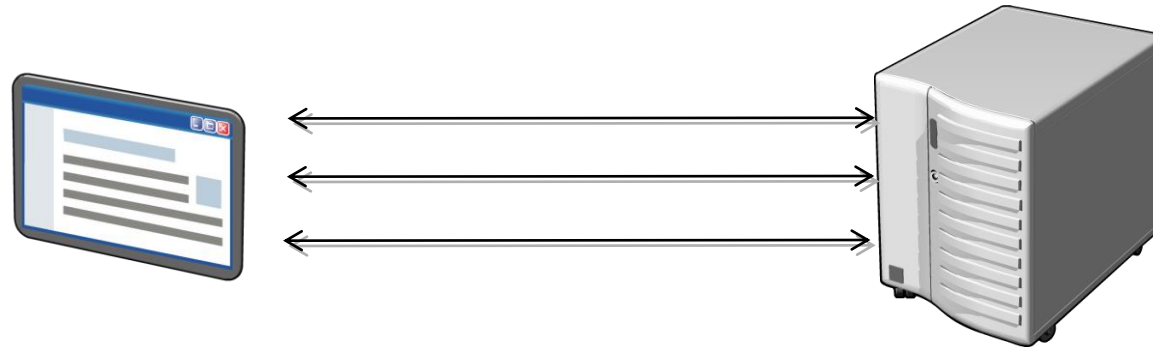
---

- › Klassische Techniken waren
  - › Long-Polling
  - › Forever-Frames
- › Mit HTML5 kamen JavaScript APIs auf, wie
  - › Server-sent Events
  - › Web Sockets
- › Die HTML5 APIs erlauben Kommunikation in Echtzeit

# WebSockets

---

- › Web Sockets erlauben bidirektionale Kommunikation in Echtzeit über das Web



- › RFC6455 definiert die ws- und wss-Protokolle
- › Die WebSockets-API definiert eine JavaScript-API für die Kommunikation mit einem WS-Server aus dem Browser

# Das WebSocket-Objekt

---

› Das WebSocket-Objekt kapselt alle Funktionen, die zur Kommunikation mit einem Socket-fähigen Server benötigt werden

› Aufbau einer neuen Verbindung:

```
let socket = new WebSocket('ws://127.0.0.1/bookings');
```

› Prüfen ob die Verbindung erfolgreich zustande kam:

```
socket.onopen = function() {  
    alert("Connection to server now open!");  
};
```

› Mit der Methode `close()` wird eine Verbindung geschlossen:

```
socket.close();
```

# Das WebSocket-Objekt – II

---

- › Die Methode `send()` schickt eine Nachricht an den Server:
  - › `let message = ...;`
  - › `socket.send(message);`
- › Über die Eigenschaft `bufferedAmount` kann festgestellt werden
  - › Ob es noch nicht versendete (gepufferte) Nachrichten gibt
  - › Ob eine Nachricht versendet wurde
- › Über das `error`-Event kann auf evtl. Fehler reagiert werden
- › Nachrichten können Texte, Binärdaten oder Arrays sein

# Das WebSocket-Objekt – III

---

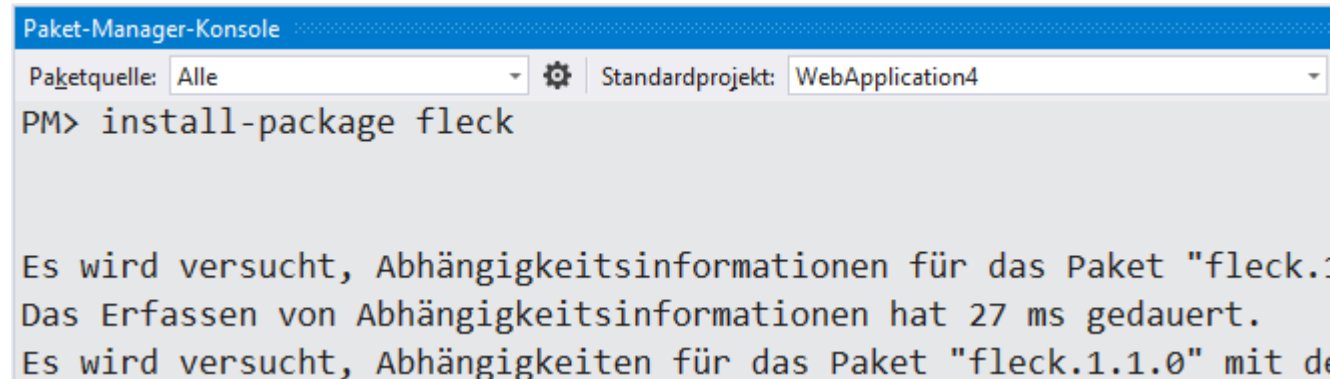
- › Das message-Event wird beim Empfang einer Nachricht ausgelöst:
  - › Die Eigenschaft `event.type` zeigt an, ob Text oder Binärdaten kommen
  - › Die Eigenschaft `event.data` enthält die Nachricht selbst

```
socket.onmessage = function(event)
{
    if (event.type == "Text") {
        handleTextMessage(event.data);
    }
    else {
        handleBinaryMessage(socket.binaryType, event.data);
    }
};
```

# Die Server-Seite

---

- › Es gibt diverse Implementierungen des WS-Protokolls auf der Server-Seite
- › Für .NET ist *Fleck* eine schlanke Implementierung
- › Im VS : Extras -> NuGet-Paket-Manager -> Paket-Manager-Konsole



```
Paket-Manager-Konsole
Paketquelle: Alle [gear icon] Standardprojekt: WebApplication4
PM> install-package fleck

Es wird versucht, Abhängigkeitsinformationen für das Paket "fleck.1.1.0" zu ermitteln.
Das Erfassen von Abhängigkeitsinformationen hat 27 ms gedauert.
Es wird versucht, Abhängigkeiten für das Paket "fleck.1.1.0" mit der Version 1.1.0 zu ermitteln.
```



# Sehr kleiner WebSocket-Service

---

```
var allSockets = new List<IWebSocketConnection>();

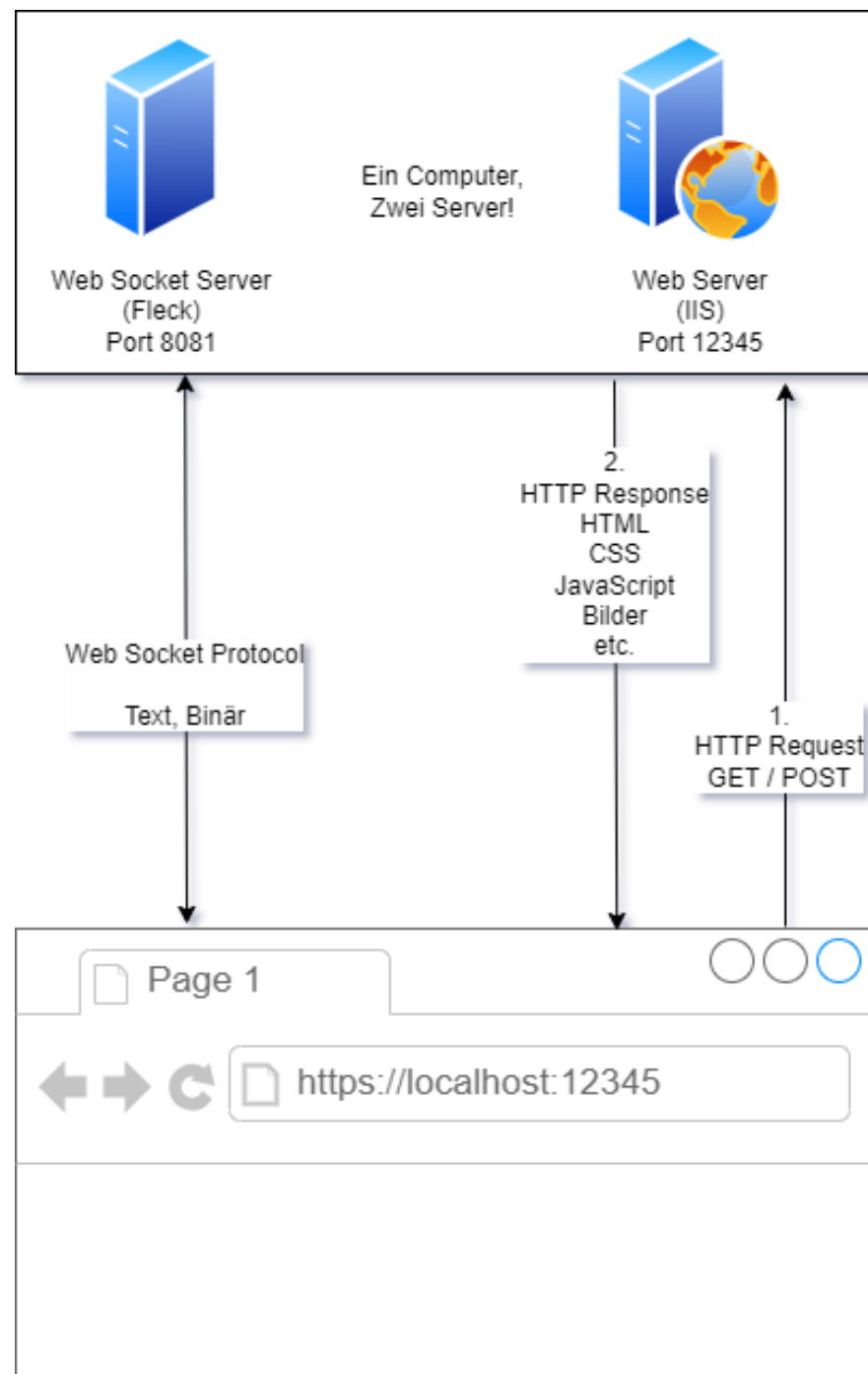
var server = new WebSocketServer("ws://127.0.0.1:8081");

server.Start(socket =>
{
    socket.OnOpen = () => allSockets.Add(socket);
    socket.OnClose = () => allSockets.Remove(socket);
    socket.OnMessage = message =>
    {
        String response = null;

        if (message == "getdate")
        {
            response = DateTime.Now.ToShortDateString();
        }

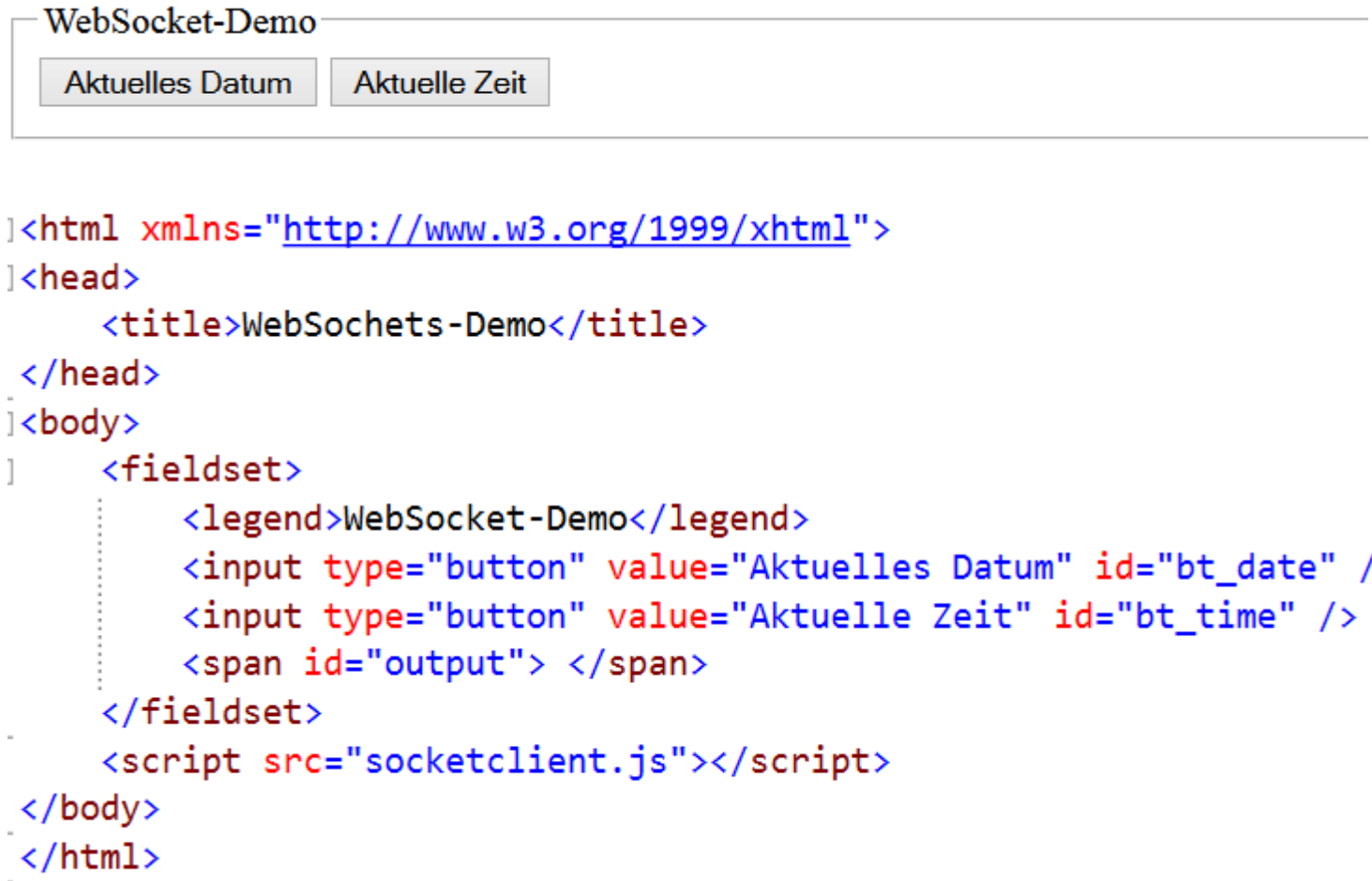
        if (message == "gettime")
        {
            response = DateTime.Now.ToLongTimeString();
        }

        foreach (var s in allSockets)
        {
            s.Send(response ?? "invalid Request");
        }
    }
});
```



# Die Sicht der Demo-Anwendung

---



# Client-seitiger Code

---

```
var socket = new WebSocket("ws://127.0.0.1:8081");

socket.onopen = function ()
{
    document.getElementById('log').innerHTML = "WebSocket erfolgreich geöffnet";
};

socket.onclose = function ()
{
    document.getElementById('log').innerHTML = "WebSocket geschlossen";
};

socket.onmessage = function (event)
{
    document.getElementById('output').innerHTML = event.data;
};
```

# Client-seitiger Code – II

---

```
socket.onerror = function (event)
{
    document.getElementById('log').innerHTML = "Fehler: " + JSON.stringify(event);
};

document.getElementById('bt_date').onclick = function ()
{
    socket.send("getdate");
};

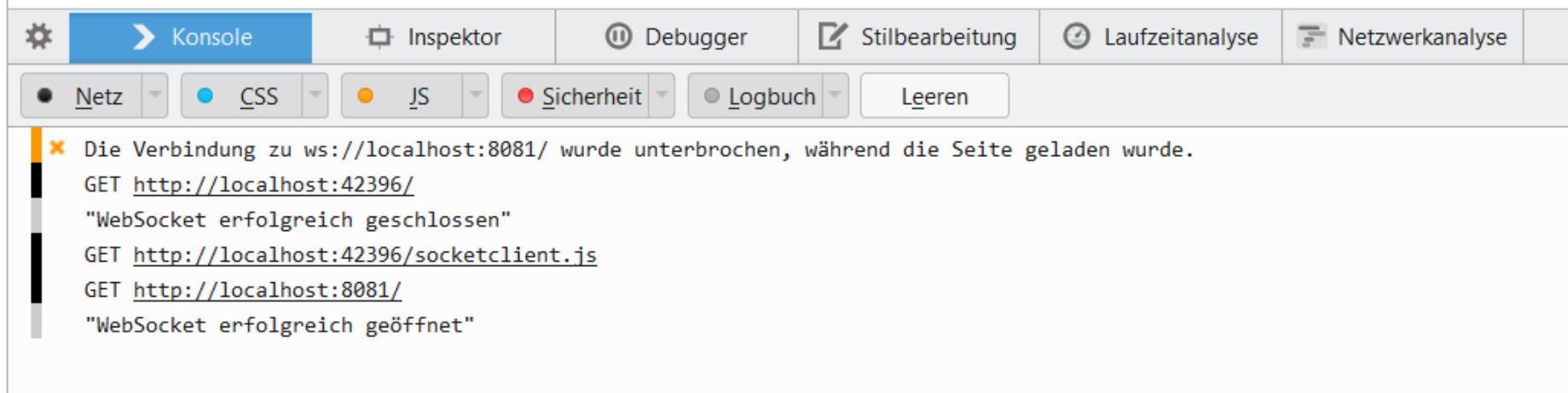
document.getElementById('bt_time').onclick = function ()
{
    socket.send("gettime");
};
```

# Start der Demo-Anwendung

---

## WebSocket-Demo

Aktuelles Datum Aktuelle Zeit 10:58:13



The screenshot shows a web browser's developer console. The top bar contains tabs for 'Konsole' (selected), 'Inspektor', 'Debugger', 'Stilbearbeitung', 'Laufzeitanalyse', and 'Netzwerkanalyse'. Below this, there are filters for 'Netz', 'CSS', 'JS', 'Sicherheit', and 'Logbuch', along with a 'Leeren' button. The console log displays the following messages:

- ✖ Die Verbindung zu ws://localhost:8081/ wurde unterbrochen, während die Seite geladen wurde.
- GET <http://localhost:42396/>
- "WebSocket erfolgreich geschlossen"
- GET <http://localhost:42396/socketclient.js>
- GET <http://localhost:8081/>
- "WebSocket erfolgreich geöffnet"