Atelier WS 07 : Server-Sent Events (SSE)

Objectif

Création d'une application de notifications en temps réel avec SSE via :

- Un serveur SSE qui émet des événements
- Un client web qui les affiche en temps réel
- Un système de notifications multiples

Setup Initial

Création du projet

```
mkdir sse-workshop
cd sse-workshop
```

Structure

```
mkdir public server
touch server/app.js public/index.html public/app.js package.json
```

package.json

```
{ "name": "sse-workshop",
  "version": "1.0.0",
  "type": "module",
  "scripts": {
    "start": "node server/app.js",
    "dev": "nodemon server/app.js"
  },
  "dependencies": {
    "express": "^4.18.0",
    "cors": "^2.8.5"
  },
  "devDependencies": {
    "nodemon": "^3.0.0"
  }
}
```

Étape 1 : Serveur SSE Complet (server/app.js)

```
import express from 'express';
import cors from 'cors';
import path from 'path';
import { fileURLToPath } from 'url';

const __filename = fileURLToPath(import.meta.url);
const __dirname = path.dirname(__filename);

const app = express();
const PORT = 3000;

// Middleware
app.use(cors());
app.use(express.json());
```

```
app.use(express.static('public'));

// Stockage des clients connectés
const clients = new Map();
let clientId = 0;

// Routes
app.get('/', (req, res) => {
  res.sendFile(path.join(__dirname, '../public/index.html'));
});

// Endpoint SSE principal
app.get('/events', (req, res) => {
  // Headers SSE obligatoires
  res.writeHead(200, {
    'Content-Type': 'text/event-stream',
    'Cache-Control': 'no-cache',
    'Connection': 'keep-alive',
    'Access-Control-Allow-Origin': '*'
  });

  // Heartbeat pour garder la connexion active
  const heartbeat = setInterval(() => {
    res.write(':\n\n'); // Commentaire SSE pour heartbeat
  }, 30000);

  // ID unique pour ce client
  const id = ++clientId;
  clients.set(id, res);

  console.log(`Client ${id} connecté - Total: ${clients.size}`);

  // Envoyer un événement de bienvenue
  sendEvent(res, 'connection', {
    id: id,
    message: 'Connecté au serveur SSE',
    timestamp: new Date().toISOString(),
    clients: clients.size
  });

  // Nettoyer à la déconnexion
  req.on('close', () => {
    clearInterval(heartbeat);
    clients.delete(id);
    console.log(`Client ${id} déconnecté - Restants: ${clients.size}`);
  });
});

// API pour envoyer des événements manuellement
app.post('/api/notify', (req, res) => {
  const { type, message, data } = req.body;

  if (!type || !message) {
```

```
    return res.status(400).json({ error: 'Type et message requis' });
  }

  broadcastEvent(type, {
    message,
    data: data || {},
    timestamp: new Date().toISOString(),
    source: 'api'
  });

  res.json({
    success: true,
    sentTo: clients.size,
    event: type
  });
});

// Simulation d'événements automatiques
function startEventSimulation() {
  const eventTypes = ['user_joined','message_received','system_alert',
    'data_updated','notification'];

  const messages = {
    user_joined: ['New utilisateur connecté', 'User a rejoint la session'],
    message_received: ['Nouveau message', 'Message privé reçu'],
    system_alert: ['Maintenance planifiée', 'Sauvegarde complétée'],
    data_updated: ['Données mises à jour', 'Sync terminée'],
    notification: ['Rappel important', 'Nouvelle fonctionnalité']
  };

  setInterval(() => {
    if (clients.size > 0) {
const type = eventTypes[Math.floor(Math.random() * eventTypes.length)];
const messageOptions = messages[type];
const message = messageOptions[Math.floor(Math.random() *
messageOptions.length)];

      broadcastEvent(type, {message,count: Math.floor(Math.random() * 100),
        timestamp: new Date().toISOString(), source: 'simulation' });
    }
  }, 8000); // Toutes les 8 secondes
}

// Fonction utilitaire pour envoyer un événement
function sendEvent(res, event, data) {
  res.write(`event: ${event}\n`);
  res.write(`data: ${JSON.stringify(data)}\n\n`);
}

// Diffuser à tous les clients
function broadcastEvent(event, data) {
  console.log(`Broadcast: ${event} to ${clients.size} clients`);
```

```
  clients.forEach((res, id) => {
    try {
      sendEvent(res, event, { ...data, clientId: id });
    } catch (error) {
      console.error(`Erreur envoi client ${id}:`, error.message);
      clients.delete(id);
    }
  });
}

// Démarrer le serveur
app.listen(PORT, () => {
  console.log(`Serveur SSE démarré sur http://localhost:${PORT}`);
  console.log(`Endpoint SSE: http://localhost:${PORT}/events`);
  console.log(`API Notify: POST http://localhost:${PORT}/api/notify`);

  startEventSimulation();
});
```

Étape 2 : Client Web Avancé (public/index.html)

```
<!DOCTYPE html>
<html lang="fr">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title> Atelier SSE - Notifications Temps Réel</title>
    <link rel="stylesheet"
href="https://cdnjs.cloudflare.com/ajax/libs/font-
awesome/6.0.0/css/all.min.css">
    <style>
        * { margin: 0;padding: 0;box-sizing: border-box;  }
        body {font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
            background: linear-gradient(135deg, #667eea 0%, #764ba2 100%);
            min-height: 100vh; padding: 20px; }
        .container {max-width: 1200px;margin: 0 auto; }
        .header { text-align: center; color: white; margin-bottom: 30px;}
        .header h1 {font-size: 2.5rem;margin-bottom: 10px; }
        .header .subtitle {font-size: 1.2rem; opacity: 0.9; }
        .dashboard { display: grid; grid-template-columns: 1fr 300px;gap:
20px; background: white; border-radius: 15px;overflow: hidden;box-shadow: 0
20px 40px rgba(0,0,0,0.1); }
        .events-panel {padding: 20px;  }
        .controls-panel {background: #f8f9fa;
            padding: 20px; border-left: 1px solid #e9ecef; }

        .connection-status {
            display: flex;
            align-items: center;
            gap: 10px;
            padding: 15px;
            background: #e8f5e8;
            border-radius: 10px;
            margin-bottom: 20px;        }
```

```
        .status-connected { background: #e8f5e8; color: #2e7d32; }
        .status-disconnected { background: #ffebee; color: #c62828; }
        .events-list { max-height: 500px; overflow-y: auto; border: 1px
solid #e9ecef; border-radius: 10px;padding: 10px;}
        .event-item { padding: 15px; margin-bottom: 10px;
            border-radius: 8px; border-left: 4px solid;
            animation: slideIn 0.3s ease-out;  }

        @keyframes slideIn {
            from { transform: translateX(-10px); opacity: 0; }
            to { transform: translateX(0); opacity: 1; }
        }

   .event-user_joined { border-color: #4caf50; background: #f1f8e9; }
   .event-message_received { border-color: #2196f3; background: #e3f2fd; }
   .event-system_alert { border-color: #ff9800; background: #fff3e0; }
   .event-data_updated { border-color: #9c27b0; background: #f3e5f5; }
   .event-notification { border-color: #607d8b; background: #eceff1; }
   .event-connection { border-color: #00bcd4; background: #e0f7fa; }

   .event-header {display: flex; justify-content: between; align-items:
center; margin-bottom: 5px;}
   .event-type {font-weight: bold; text-transform: uppercase; font-size:
0.8rem;      }
   .event-time {color: #666; font-size: 0.8rem;       }
   .event-message {margin: 5px 0;  }
   .event-data { font-family: monospace; font-size: 0.8rem; color: #666;
margin-top: 5px; }
        .stats { display: grid; grid-template-columns: 1fr 1fr;
            gap: 10px; margin-bottom: 20px;       }
        .stat-card {
            background: white;
            padding: 15px;
            border-radius: 10px;
            text-align: center;
            box-shadow: 0 2px 5px rgba(0,0,0,0.1);
        }
       .stat-number {font-size: 1.5rem; font-weight: bold;display: block; }
       .stat-label { font-size: 0.8rem; color: #666; }
       .control-group { margin-bottom: 20px; }
       .control-group h3 { margin-bottom: 10px; color: #333;}

        button {
            width: 100%;
            padding: 12px;
            border: none;
            border-radius: 8px;
            background: #667eea;
            color: white;
            font-size: 1rem;
            cursor: pointer;
            transition: background 0.3s;
            margin-bottom: 10px;        }
```

```
        button:hover {background: #5a6fd8;  }
        button:disabled { background: #ccc;  cursor: not-allowed; }
        .form-group {     margin-bottom: 15px;     }
        label { display: block;  margin-bottom: 5px; font-weight: 500; }

        input, select, textarea {
            width: 100%;
            padding: 10px;
            border: 1px solid #ddd;
            border-radius: 5px;
            font-size: 1rem;        }
        textarea { height: 80px; resize: vertical; }
        .notification-badge {
            position: fixed;
            top: 20px;
            right: 20px;
            background: #ff4757;
            color: white;
            padding: 10px 15px;
            border-radius: 20px;
            animation: bounce 0.5s;        }
        @keyframes bounce {
            0%, 20%, 60%, 100% { transform: translateY(0); }
            40% { transform: translateY(-10px); }
            80% { transform: translateY(-5px); }
        }
    </style>
</head>
<body>
<div class="container">
 <div class="header">
  <h1><i class="fas fa-bolt"></i> Atelier SSE</h1>
     <div class="subtitle">Notifications en Temps Réel</div>
  </div>

<div class="dashboard">
  <div class="events-panel">
  <div id="connectionStatus" class="connection-status status-disconnected">
    <i class="fas fa-plug"></i>
      <span>Déconnecté</span>
  </div>

  <div class="stats">
    <div class="stat-card">
      <span id="eventsCount" class="stat-number">0</span>
       <span class="stat-label">Événements</span>
     </div>
<div class="stat-card">
   <span id="clientsCount" class="stat-number">0</span>
         <span class="stat-label">Clients connectés</span>
 </div></div>

 <h3>Événements en Temps Réel</h3>
```

```
<div id="eventsList" class="events-list">
 <div class="event-item event-connection">
  <div class="event-message">En attente de connexion...</div>
  </div></div></div>
<div class="controls-panel">
   <div class="control-group">
     <h3><i class="fas fa-cog"></i> Contrôles</h3>
        <button id="connectBtn" onclick="connectSSE()">
          <i class="fas fa-plug"></i> Se connecter
        </button>
        <button id="disconnectBtn" onclick="disconnectSSE()" disabled>
           <i class="fas fa-power-off"></i> Se déconnecter
       </button> </div>
    <div class="control-group">
      <h3><i class="fas fa-paper-plane"></i> Envoyer une notification</h3>
         <div class="form-group">
           <label for="eventType">Type d'événement:</label>
            <select id="eventType">
             <option value="notification">Notification</option>
             <option value="user_joined">User Joined</option>
             <option value="message_received">Message Received</option>
             <option value="system_alert">System Alert</option>
             <option value="data_updated">Data Updated</option>
            </select></div>
    <div class="form-group">
      <label for="eventMessage">Message:</label>
     <textarea id="eventMessage" placeholder="Entrez message"></textarea>
    </div>
    <button onclick="sendCustomEvent()">
       <i class="fas fa-paper-plane"></i> Envoyer
    </button>
    </div>
    <div class="control-group">
      <h3><i class="fas fa-info-circle"></i> Informations</h3>
        <p style="font-size: 0.9rem; color: #666;">
 Les événements automatiques sont envoyés toutes les 8 secondes.
 Utilisez les contrôles pour envoyer des notifications personnalisées.
</p></div></div> </div> </div>
<script src="app.js"></script>
</body>
</html>
```

Étape 3 : JavaScript Client (public/app.js)

```
class SSEClient {
    constructor() {
        this.eventSource = null;
        this.isConnected = false;
        this.eventsCount = 0;

        this.initializeElements();
        this.setupEventListeners();
    }
```

```
    initializeElements() {
        this.connectionStatus =
document.getElementById('connectionStatus');
        this.connectBtn = document.getElementById('connectBtn');
        this.disconnectBtn = document.getElementById('disconnectBtn');
        this.eventsList = document.getElementById('eventsList');
        this.eventsCountEl = document.getElementById('eventsCount');
        this.clientsCountEl = document.getElementById('clientsCount');
    }

    setupEventListeners() {
        // Reconnexion automatique si déconnecté
        window.addEventListener('online', () => {
            if (!this.isConnected) {   this.connect(); }
        });
    }

    connect() {
        if (this.isConnected) return;
        try {
            this.eventSource = new EventSource('/events');
            this.eventSource.onopen = (event) => {
             this.handleConnectionChange(true, 'Connecté au serveur SSE');
            };

            this.eventSource.onmessage = (event) => {
                console.log('Message sans événement:', event.data);
            };

            // Gestionnaire d'événements générique
            this.eventSource.onerror = (error) => {
                console.error('Erreur SSE:', error);
                this.handleConnectionChange(false, 'Erreur de connexion');
                this.eventSource.close();
            };

            // Événements spécifiques
            this.setupEventHandlers();

        } catch (error) {
            console.error('Erreur création EventSource:', error);
            this.handleConnectionChange(false, 'Erreur de connexion');
        }}

    setupEventHandlers() {
        const events = ['connection', 'user_joined', 'message_received',
 'system_alert', 'data_updated', 'notification'];

        events.forEach(eventType => {
            this.eventSource.addEventListener(eventType, (event) => {
                this.handleEvent(eventType, JSON.parse(event.data));
            });
        });
```

```
    }

    handleEvent(type, data) {
        this.eventsCount++;
        this.eventsCountEl.textContent = this.eventsCount;

        // Mettre à jour le compteur de clients si disponible
        if (data.clients !== undefined) {
            this.clientsCountEl.textContent = data.clients;}

        this.addEventToList(type, data);
        this.showNotification(type, data.message); }

    addEventToList(type, data) {
        const eventItem = document.createElement('div');
        eventItem.className = `event-item event-${type}`;

        const time = new Date(data.timestamp).toLocaleTimeString();

        eventItem.innerHTML = `<div class="event-header"><span
class="event-type">${type}</span><span class="event-
time">${time}</span></div><div class="event-
message">${data.message}</div><div class="event-data">Client:
${data.clientId||'N/A'}|Source: ${data.source || 'unknown'}${data.count ? `
| Count: ${data.count}` : ''}</div> `;

        this.eventsList.prepend(eventItem);

        // Limiter à 50 événements maximum
        if (this.eventsList.children.length > 50) {
            this.eventsList.removeChild(this.eventsList.lastChild); }
    }

    showNotification(type, message) {
        // Créer une notification toast
        const notification = document.createElement('div');
        notification.className = 'notification-badge';
        notification.innerHTML =
`<strong>${type.toUpperCase()}</strong><br> ${message} `;

        document.body.appendChild(notification);

        // Supprimer après 3 secondes
        setTimeout(() => {
            if (notification.parentNode) {
                notification.parentNode.removeChild(notification);}
        }, 3000);
    }

    handleConnectionChange(connected, message) {
        this.isConnected = connected;
```

```
        this.connectionStatus.className = ‘connection-status status-
${connected ? 'connected' : 'disconnected'}’;
        this.connectionStatus.innerHTML = ‘<i class="fas fa-${connected ?
'check-circle' : 'times-circle'}"></i><span>${message}</span>’;

        this.connectBtn.disabled = connected;
        this.disconnectBtn.disabled = !connected;
        if (!connected) { this.clientsCountEl.textContent = '0'; }
    }
    disconnect() {
        if (this.eventSource) {
            this.eventSource.close();
            this.eventSource = null; }
        this.handleConnectionChange(false, 'Déconnecté');}
}

// Initialisation globale
const sseClient = new SSEClient();

// Fonctions globales pour les boutons
function connectSSE() { sseClient.connect();}
function disconnectSSE() {sseClient.disconnect();}

async function sendCustomEvent() {
    const type = document.getElementById('eventType').value;
    const message = document.getElementById('eventMessage').value;

    if (!message.trim()) { alert('Veuillez entrer un message'); return; }
    try {
        const response = await fetch('/api/notify', { method: 'POST',
            headers: {'Content-Type': 'application/json',
            }, body: JSON.stringify({type: type, message: message, data: {
                    custom: true, sentAt: new Date().toISOString() }
            })
        });
        const result = await response.json();
        if (result.success) {
            document.getElementById('eventMessage').value = '';
            console.log('Notification envoyée:', result);
        } else {  alert('Erreur lors de l\'envoi'); }
    } catch (error) {
        console.error('Erreur envoi:', error);
        alert('Erreur de connexion au serveur'); }
}

// Connexion automatique au chargement
window.addEventListener('load', () => { setTimeout(connectSSE, 1000);});
```

Étape 4 : Lancement et Test

```
# Installer les dépendances
npm install

# Démarrer le serveur
npm run dev
```

Aller sur : http://localhost:3000

Test 1 : Connexion SSE

```
// Dans la console navigateur

fetch('http://localhost:3000/events')

  .then(response => console.log('Statut:', response.status))
```

Test 2 : Envoi de Notification

```
// Via l'interface ou curl
curl -X POST http://localhost:3000/api/notify \
  -H "Content-Type: application/json" \
  -d '{"type": "system_alert","message": "Maintenance planifiée à minuit"
  }'
```