DAVIDE PISANÒ - ANTONIO SERVETTI

IDENTIFICAZIONE AUDIO RESISTENTE AL RUMORE IN WEBASSEMBLY

Indice

Introduzione

Negli ultimi anni si è notato un trend sempre crescente nell'utilizzo di JavaScript per la creazione di applicazioni desktop.

Ci sono diversi fattori che hanno contribuito alla popolarità di JavaScript, primo fra tutti è che rappresenta il linguaggio standard, de facto, per l'implementazione di funzionalità dinamiche su pagine web. Nello specifico, JavaScript è l'unico linguaggio di scripting supportato nativamente da tutti i browser web moderni.

Ci sono stati dei tentativi per introdurre alcune novità in questo ambito, seguendo principalmente due approcci:

1. l'inclusione di una nuova macchina virtuale all'interno di un browser web che supportasse un nuovo linguaggio

2. la realizzazione di un nuovo linguaggio ma eseguito sulla stessa macchina virtuale JavaScript già presente in un web browser

Ricade nella prima categoria VBScript di Microsoft, basato su Visual Basic, introdotto a metà degli anni '90, oggi non più supportato da nessun browser moderno.

Nella seconda categoria possiamo annoverare, più recentemente, TypeScript (sempre di Microsoft), CoffeeScript e Dart (di Google). Questi linguaggi sono basati su un cosiddetto *transpiler*, ovvero un compilatore che prende in input il codice sorgente scritto ad esempio in TypeScript e lo converte in codice JavaScript, mantenendo le stesse funzionalità del codice originale.

Un altro punto di forza di JavaScript è la sua facilità di apprendimento, soprattutto rispetto ad altri linguaggi di programmazione più a basso livello come C o C++, permettendo agli sviluppatori di iniziare a scrivere codice più rapidamente, senza dover investire troppo tempo nell'apprendimento di una nuova tecnologia. Oggi, infatti, si assiste ad una quantità sempre crescente di librerie e framework per JavaScript, atti a semplificare lo sviluppo di applicazioni web complesse e a migliorarne la qualità, giusto per citarne alcuni: React, Angular, Vue.js, ma anche il più "anziano" jQuery.

Per questi ed altri motivi, durante la fine degli anni 2000, ci si è iniziati a porre una domanda: "è possibile eseguire codice JavaScript al di fuori del contesto web browser?". La risposta (affermativa) a questa domanda è stata la nascita di Node.js che ha dato il via al paradigma del *JavaScript everywhere*[1]. Questo vuol dire, in estrema sintesi, poter utilizzare lo stesso linguaggio per creare applicazioni web, sia lato front-end sia lato back-end. In teoria si potrebbe così ridurre il tempo necessario per il processo di sviluppo di un'applicazione, riducendo il portfolio di tecnologie che un programmatore deve conoscere.

[1] Del quale non mancano i detrattori

Nei primi anni 2010, quando Node.js iniziava a prendere piede, ci si è posti un'altra domanda "è possibile scrivere e distribuire applicazioni desktop/mobile scritte in JavaScript?". La risposta, ancora una volta è stata affermativa. Nasce Electron: un framework open-source che consente agli sviluppatori di creare applicazioni desktop multipiattaforma utilizzando tecnologie web standard come HTML, CSS e JavaScript. Dal lato mobile nascono tecnologie analoghe a Electron come Ionic, React Native e PhoneGap, tutte con obiettivi abbastanza simili. Man mano l'ecosistema JavaScript ha iniziato a diventare quello che Java era nei primi anni 2000 per la scrittura di applicazioni desktop consumer multipiattaforma.

Il motivo principale dietro alla popolarità di questo ecosistema basato su JavaScript è la possibilità di utilizzare un'unica codebase (in JavaScript) che può essere eseguita su piattaforme molto diverse tra loro, problematica che è molto sentita nell'ambito mobile dove si hanno due piattaforme completamente diverse: Android e iOS[2].

[2] All'epoca della prima apparizione di queste tecnologie bisognava supportare anche Windows Phone

Il trend di scrivere applicazioni in JavaScript è stato amplificato dalla crescente importanza del web come piattaforma per la distribuzione di applicazioni software. Software utilizzati quotidianamente da miliardi di utenti sono basati sul web e, per forza di cose, devono essere scritti in JavaScript.

Da qui la nascita delle cosiddette *Rich Internet Applications* (RIA), ovvero applicazioni web che offrono un'esperienza utente interattiva e avanzata simile a quella di un'applicazione desktop tradizionale. Le RIA sono caratterizzate da una vasta gamma di funzionalità e interattività che le distinguono dalla semplici pagine web statiche. Oltre alla classica triade HTML + CSS + JavaScript le RIA possono fare uso di tecnologie più avanzate e recenti come WebSockets, WebAudio, WebAssembly, WebRTC, WebVR, WebGPU, Web Animations API. In sostanza il browser diventa un'interfaccia o un'astrazione della macchina sottostante, alla quale si può accedere utilizzando JavaScript.

Nello specifico WebAudio è un'API JavaScript avanzata che consente di manipolare e generare audio all'interno del browser. È stata

progettata per consentire agli sviluppatori di creare RIA che includono funzionalità audio, come la registrazione, la riproduzione e l'elaborazione di suoni. L'API è basata su un'architettura a nodi, dove ogni nodo rappresenta una singola operazione di elaborazione del suono. I nodi possono essere collegati tra loro per creare una catena di elaborazione, in cui il suono viene elaborato in successione da ogni nodo che attraversa. La manipolazione del suono avviene in real time. WebAudio è oggi supportato su tutti i browser recenti basati sugli engine JavaScript V8 e SpiderMonkey.

La necessità di scrivere applicazioni real time ha portato la necessità di dover eseguire codice ad alta efficienza, obiettivo non realizzabile completamente con un linguaggio interpretato quale JavaScript. Alla fine degli anni 2010 nasce quindi *WebAssembly* (Wasm): un formato di codice binario portabile che consente di eseguire codice di basso livello all'interno del browser web. È stato progettato per essere compatibile con i linguaggi di programmazione come C, C++ e Rust. In pratica, quindi, Wasm permette di creare applicazioni web che eseguono codice più velocemente e con maggiore efficienza rispetto a soluzioni basate su JavaScript. Wasm è stato pensato per essere altamente interoperabile con JavaScript: codice JavaScript può richiamare codice Wasm e viceversa, creando quindi soluzioni ibride che combinano il meglio di entrambi i mondi.

Sfruttando tutte queste tecnologie e un ecosistema ormai maturo, l'obiettivo di questa tesi è quello di discutere la realizzazione di un sistema per l'identificazione di audio: un utente sottopone uno spezzone di un brano audio di pochi secondi al sistema, il quale risponde col nome di quel brano. L'obiettivo principale è quello di eseguire l'algoritmo di identificazione su dispositivi eterogenei all'interno di un web browser, utilizzando Wasm e WebAudio. Questo presenta numerosi vantaggi, tra i più importanti si possono individuare:

- l'evitare all'utente il download di un'app addizionale, potendo sfruttare le funzionalità dell'algoritmo di riconoscimento direttamente dal suo web browser
- la notevole riduzione del carico lato server: grazie alla sua architettura distribuita parte della complessità viene spostata sul dispositivo dell'utente finale, il quale porta a termine buona parte del processo di identificazione. Questo rende possibile un riconoscimento musicale più veloce ed efficiente rispetto ad altre applicazioni simili

In definitiva, si renderà l'esperienza dell'utente ancora più piacevole e soddisfacente, mantenendo le stesse funzionalità e caratteristiche di una classica app eseguita nativamente su un dispositivo dell'utente.