Báo cáo Kiến trúc Chuyên sâu: Triển khai Hệ thống Giả lập EmulatorJS Phân tán trên Hạ tầng Serverless với Cơ chế Đồng bộ Hóa Dữ liệu Thời gian thực

1. Tổng quan Điều hành và Phạm vi Nghiên cứu

Sự chuyển dịch của công nghệ giả lập (emulation) từ các ứng dụng máy tính để bàn (desktop native applications) sang môi trường trình duyệt web (browser-based) đại diện cho một bước tiến quan trọng trong việc bảo tồn di sản phần mềm và dân chủ hóa khả năng truy cập nội dung số. Dự án triển khai **EmulatorJS** trên nền tảng **GitHub Pages** với kho lưu trữ ROM nội bộ và đồng bộ hóa dữ liệu trò chơi (Save Game) thông qua **Firebase Realtime Database** không chỉ đơn thuần là việc tích hợp các thư viện JavaScript, mà là một bài toán phức tạp về kiến trúc hệ thống phân tán, quản lý bộ nhớ WebAssembly (Wasm) và bảo mật ứng dụng web hiện đại.

Báo cáo này cung cấp một phân tích toàn diện và lộ trình kỹ thuật chi tiết nhằm xây dựng một hệ thống giả lập hoàn chỉnh, hoạt động hoàn toàn trên phía máy khách (client-side) mà không phụ thuộc vào máy chủ ứng dụng truyền thống (backend server). Kiến trúc được đề xuất tận dụng sức mạnh của mô hình Serverless để giảm thiểu chi phí vận hành, đồng thời đảm bảo trải nghiệm người dùng liền mạch thông qua việc đồng bộ hóa dữ liệu trạng thái trò chơi (Save State/Save RAM) giữa các thiết bị khác nhau.

Phạm vi nghiên cứu bao gồm việc phân tích sâu về cơ chế hoạt động của lõi Libretro trong môi trường Emscripten, kỹ thuật vượt qua các rào cản bảo mật Cross-Origin trên GitHub Pages để kích hoạt đa luồng (multithreading), và chiến lược quản lý dữ liệu nhị phân (binary data) hiệu quả trên cơ sở dữ liệu NoSQL thời gian thực. Đặc biệt, báo cáo sẽ tập trung giải quyết thách thức cốt lõi: làm thế nào để can thiệp vào hệ thống tệp tin ảo (Virtual File System) của trình giả lập để "tiêm" (inject) và "trích xuất" (extract) dữ liệu save game một cách lập trình mà không làm gián đoạn trải nghiệm người dùng.[1]

---

2. Cơ sở Lý thuyết và Phân tích Nền tảng Công nghệ

Để triển khai thành công hệ thống phức tạp này, việc thấu hiểu sâu sắc các thành phần công nghệ nền tảng và sự tương tác giữa chúng là điều kiện tiên quyết.

2.1. EmulatorJS và Kiến trúc Libretro trên Web

EmulatorJS thực chất là một lớp bao bọc (wrapper) thông minh xung quanh dự án RetroArch và hệ sinh thái Libretro, được biên dịch sang WebAssembly để chạy trên trình duyệt.

Kiến trúc Đa Lõi (Multi-Core Architecture):
Khác với các trình giả lập đơn lẻ, EmulatorJS sử dụng API Libretro để giao tiếp với các "lõi" (cores) giả lập. Mỗi hệ máy (như NES, SNES, GBA, PS1) tương ứng với một lõi riêng biệt (ví dụ: fceumm cho NES, snes9x cho SNES). Các lõi này được viết bằng C/C++ và biên dịch sang .wasm và .js thông qua chuỗi công cụ Emscripten. Điều này có nghĩa là logic giả lập CPU, GPU và xử lý âm thanh diễn ra ở tầng thấp (low-level) với hiệu suất gần như native, trong khi giao diện người dùng và quản lý file được xử lý bởi JavaScript.2
Hệ thống Tệp Ảo (Emscripten Virtual File System - MEMFS):
Đây là yếu tố quan trọng nhất đối với yêu cầu "đồng bộ Save Game". Vì lý do bảo mật, mã Wasm chạy trong trình duyệt không thể truy cập trực tiếp ổ cứng người dùng. Emscripten tạo ra một hệ thống tệp ảo nằm trong bộ nhớ RAM (MEMFS). Khi một trò chơi thực hiện lệnh ghi save (ví dụ: ghi vào SRAM), nó thực chất đang ghi vào một vùng nhớ trong RAM của trình duyệt tại đường dẫn ảo, thường là /home/web_user/retroarch/userdata/saves/.4 Dữ liệu này sẽ mất đi khi người dùng tải lại trang nếu không có cơ chế can thiệp để trích xuất và lưu trữ bền vững.

2.2. Thách thức Môi trường Hosting GitHub Pages

GitHub Pages là một giải pháp hosting tĩnh tuyệt vời, nhưng nó đi kèm với những hạn chế kỹ thuật nghiêm trọng đối với các ứng dụng WebAssembly hiệu năng cao.

Vấn đề SharedArrayBuffer và Bảo mật:
Để đạt hiệu suất tối ưu, đặc biệt là với các hệ máy hiện đại hơn, các lõi giả lập cần sử dụng đa luồng (multithreading), dựa trên tính năng SharedArrayBuffer của JavaScript. Tuy nhiên, do các lỗ hổng bảo mật vi xử lý (như Spectre và Meltdown), các trình duyệt hiện đại đã vô hiệu hóa SharedArrayBuffer trừ khi trang web được phục vụ trong một ngữ cảnh bảo mật cô lập (Cross-Origin Isolated context). Điều này yêu cầu máy chủ phải gửi kèm hai HTTP Headers đặc biệt:

1. Cross-Origin-Embedder-Policy (COEP): require-corp
2. Cross-Origin-Opener-Policy (COOP): same-origin

GitHub Pages không cho phép người dùng tùy chỉnh HTTP Headers của máy chủ. Đây là một rào cản kỹ thuật lớn. Nếu không giải quyết vấn đề này, EmulatorJS sẽ hoạt động ở chế độ đơn luồng, gây giật lag hoặc không thể tải các lõi nặng.[6]

2.3. Firebase Realtime Database và Xử lý Dữ liệu Nhị phân

Firebase Realtime Database (RTDB) là một cơ sở dữ liệu NoSQL lưu trữ dữ liệu dưới dạng cây JSON và đồng bộ hóa thời gian thực tới mọi client được kết nối.

Sự không tương thích dữ liệu:
File save game (.srm, .state) là các chuỗi byte nhị phân (binary blobs). JSON và Firebase RTDB

không hỗ trợ lưu trữ dữ liệu nhị phân trực tiếp. Do đó, một cơ chế chuyển đổi (encoding/decoding) là bắt buộc. Phương pháp phổ biến và tương thích nhất là mã hóa dữ liệu nhị phân thành chuỗi Base64 trước khi tải lên (upload) và giải mã ngược lại khi tải xuống (download).9

Đồng bộ Thời gian thực:

Tính năng lắng nghe sự kiện (event listeners) của RTDB cho phép ứng dụng web phản ứng ngay lập tức khi dữ liệu save trên đám mây thay đổi. Điều này tạo ra khả năng đồng bộ hóa liền mạch: người dùng lưu game trên điện thoại, và khi mở máy tính, save game đó đã sẵn sàng để tải xuống.

---

3. Kiến trúc Giải pháp Tổng thể

Giải pháp được thiết kế theo mô hình **Client-Serverless**, trong đó trình duyệt web đóng vai trò trung tâm điều phối mọi hoạt động, từ giả lập đến quản lý dữ liệu.

3.1. Sơ đồ Luồng Dữ liệu (Data Flow Architecture)

Hệ thống hoạt động dựa trên các luồng dữ liệu chính sau:

1. **Luồng Khởi tạo (Initialization Flow):**
   - Trình duyệt tải trang từ GitHub Pages.
   - coi-serviceworker kích hoạt, reload trang để thiết lập môi trường bảo mật COOP/COEP.
   - Ứng dụng kết nối tới Firebase Auth để xác thực người dùng.
2. **Luồng Tải Game và Inject Save (Load & Inject Flow):**
   - Người dùng chọn ROM từ danh sách nội bộ.
   - Hệ thống truy vấn Firebase RTDB để lấy chuỗi Base64 của save game tương ứng.
   - Chuỗi Base64 được giải mã thành Uint8Array.
   - Dữ liệu được ghi trực tiếp vào hệ thống tệp ảo của Emscripten (Module.FS) tại đường dẫn định trước *trước khi* hoặc *ngay khi* lõi giả lập khởi động.
3. **Luồng Chơi và Đồng bộ Save (Gameplay & Sync Flow):**
   - Lõi giả lập đọc/ghi file save trong bộ nhớ ảo trong quá trình chơi.
   - Hệ thống lắng nghe sự thay đổi của file save (thông qua hook EJS_onSaveUpdate hoặc polling).
   - Khi có thay đổi, file save được đọc từ bộ nhớ ảo, mã hóa thành Base64 và đẩy lên Firebase RTDB.

3.2. Cấu trúc Dự án Đề xuất

Để đảm bảo tính tổ chức và khả năng bảo trì, cấu trúc thư mục trên GitHub Repository nên được sắp xếp khoa học:

Đường dẫn / Tệp tin	Mô tả Chức năng
/index.html	Điểm vào chính của ứng dụng, chứa container cho EmulatorJS.
/coi-serviceworker.js	Script quan trọng để xử lý headers bảo mật cho GitHub Pages.[12]
/js/main.js	Logic chính: điều phối UI, EmulatorJS và Firebase.
/js/firebase-config.js	Cấu hình và khởi tạo Firebase SDK (V9 Modular).
/data/	Thư mục chứa tài nguyên EmulatorJS (loader.js, các file.wasm,.js của core).
/roms/	Kho lưu trữ ROM nội bộ, phân chia theo hệ máy (ví dụ: /roms/nes/, /roms/gba/).
/assets/	Hình ảnh bìa game, CSS styles.
/gamelist.json	Tệp cấu hình định nghĩa danh sách game, đường dẫn ROM và metadata.

---

4. Triển khai Chi tiết: Kỹ thuật Cross-Origin Isolation trên GitHub Pages

Việc đầu tiên và quan trọng nhất để đảm bảo hiệu năng của EmulatorJS là giải quyết vấn đề headers bảo mật. Nếu không có bước này, các tính năng nâng cao và độ ổn định của giả lập sẽ bị ảnh hưởng nghiêm trọng.

4.1. Cơ chế của coi-serviceworker

Thư viện coi-serviceworker hoạt động như một proxy trung gian phía client (Man-in-the-Middle) ngay trong trình duyệt.

- **Bước 1:** Khi người dùng truy cập lần đầu, trang web chưa có headers bảo mật. Script coi-serviceworker.js phát hiện điều này và đăng ký chính nó làm Service Worker.

- **Bước 2:** Script tự động tải lại trang (window.reload).
- **Bước 3:** Ở lần tải thứ hai, Service Worker chặn tất cả các yêu cầu mạng (network requests) đi và về. Nó bổ sung headers Cross-Origin-Embedder-Policy: require-corp và Cross-Origin-Opener-Policy: same-origin vào phản hồi (response) trước khi trả về cho trình duyệt.
- **Kết quả:** Trình duyệt nhận thấy headers hợp lệ và kích hoạt môi trường an toàn, cho phép sử dụng SharedArrayBuffer.[12]

4.2. Mã nguồn Triển khai coi-serviceworker

Tải tệp coi-serviceworker.js từ kho lưu trữ chính thức và đặt tại thư mục gốc (root) của repository. Trong tệp index.html, script này phải được tham chiếu ở vị trí cao nhất có thể trong thẻ <head>, trước bất kỳ script nào khác.

HTML

```
<!DOCTYPE html>
<html lang="vi">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>RetroCloud Emulator</title>
    <script src="coi-serviceworker.js"></script>
    <link rel="stylesheet" href="style.css">
</head>
<body>
    </body>
</html>
```

Lưu ý: Script này không hoạt động nếu người dùng mở file HTML trực tiếp từ ổ cứng (file:// protocol) mà phải thông qua giao thức HTTP/HTTPS (như GitHub Pages hoặc local server).[14]

---

5. Thiết lập Hệ thống ROM Lưu Nội bộ (Internal ROM Storage)

Yêu cầu "ROM lưu nội bộ" đòi hỏi một chiến lược quản lý tài sản số (digital asset management) ngay trong cấu trúc của ứng dụng web tĩnh.

5.1. Quản lý Metadata qua JSON

Thay vì mã hóa cứng (hard-code) đường dẫn ROM trong mã JavaScript, việc sử dụng một tệp gamelist.json giúp tách biệt dữ liệu và logic, dễ dàng mở rộng thêm game mới mà không cần sửa code.

Mẫu gamelist.json:

JSON

Trong cấu trúc này, id đóng vai trò là khóa chính (primary key) để định danh file save trên Firebase. rom_path là đường dẫn tương đối tính từ thư mục gốc của GitHub Pages.[15]

5.2. Logic Tải ROM

Khi người dùng chọn một game từ giao diện, ứng dụng sẽ đọc thông tin từ JSON và cấu hình biến môi trường cho EmulatorJS. Vì ROM được lưu nội bộ, ta có thể truyền trực tiếp đường dẫn URL tương đối vào biến EJS_gameUrl.

JavaScript

```
function selectGame(gameId) {
  const game = gameList.find(g => g.id === gameId);
  if (!game) return;

  window.EJS_core = game.core;
  window.EJS_gameUrl = game.rom_path; // Load trực tiếp từ GitHub Pages
  window.EJS_pathtodata = 'data/';

  // Kích hoạt quy trình đồng bộ hóa trước khi start game
  initSyncAndStart(game);
}
```

---

6. Chiến lược Tích hợp Firebase Realtime Database

Phần này phân tích sâu về cách tổ chức dữ liệu và bảo mật trên Firebase để phục vụ mục đích lưu trữ save game.

6.1. Mô hình Dữ liệu (Data Modeling)

Cấu trúc dữ liệu phẳng (flattened data structure) được khuyến nghị cho Realtime Database để tối ưu hóa hiệu suất đọc/ghi. Dữ liệu save game nên được tổ chức theo User ID (UID) để đảm bảo tính riêng tư.

Cấu trúc cây JSON đề xuất:

JSON

```
{
  "users": {
    "UID_NGUOI_DUNG_A": {
      "saves": {
        "mario_bros_nes": {
          "srm_data": "BASE64_ENCODED_STRING...",
          "timestamp": 1678892345000,
          "checksum": "md5_hash_value"
        },
        "pokemon_fire_red": {
          "srm_data": "BASE64_ENCODED_STRING...",
          "timestamp": 1678999999000
        }
      },
      "settings": {
        "auto_save_interval": 60
      }
    }
  }
}
```

- srm_data: Chứa nội dung file save đã mã hóa Base64.
- timestamp: Thời điểm lưu file, dùng để giải quyết xung đột (conflict resolution) khi đồng bộ giữa nhiều thiết bị (last-write-wins).

6.2. Cấu hình Bảo mật (Firebase Security Rules)

Vì mã nguồn frontend là công khai, việc bảo mật database là cực kỳ quan trọng. Ta cần thiết lập quy tắc sao cho người dùng chỉ có thể đọc/ghi dữ liệu của chính mình.[16]

JavaScript

```
{
  "rules": {
    "users": {
      "$uid": {
        // Chỉ cho phép truy cập nếu UID của request khớp với node dữ liệu
        ".read": "$uid === auth.uid",
        ".write": "$uid === auth.uid",
        "saves": {
          "$game_id": {
            // Validate dữ liệu: srm_data phải là chuỗi (Base64)
            ".validate": "newData.hasChildren(['srm_data', 'timestamp']) &&
newData.child('srm_data').isString()"
          }
        }
      }
    }
  }
}
```

6.3. Khởi tạo Firebase SDK

Sử dụng phiên bản Firebase SDK V9 (Modular) cho phép Tree Shaking, giúp giảm dung lượng tải trang.

JavaScript

```
import { initializeApp } from "https://www.gstatic.com/firebasejs/9.22.0/firebase-app.js";
import { getDatabase, ref, set, get, child } from
"https://www.gstatic.com/firebasejs/9.22.0/firebase-database.js";
import { getAuth, GoogleAuthProvider } from
```

```
"https://www.gstatic.com/firebasejs/9.22.0/firebase-auth.js";

const firebaseConfig = {
  // Điền thông tin config từ Firebase Console
  apiKey: "...",
  authDomain: "...",
  databaseURL: "...",
  projectId: "...",
  storageBucket: "...",
  messagingSenderId: "...",
  appId: "..."
};

const app = initializeApp(firebaseConfig);
export const db = getDatabase(app);
export const auth = getAuth(app);
export const provider = new GoogleAuthProvider();
```

---

7. Kỹ thuật Tiêm (Inject) và Trích xuất (Extract) Save Game

Đây là phần cốt lõi của giải pháp, đòi hỏi sự can thiệp sâu vào vòng đời (lifecycle) của EmulatorJS và hệ thống tệp Emscripten.

7.1. Phân tích Đường dẫn Save của Core

Mỗi core giả lập có thể định nghĩa phần mở rộng file save khác nhau. RetroArch thường chuẩn hóa về .srm (Save RAM), nhưng một số core có thể dùng .sav hoặc định dạng khác. Việc xác định đúng tên file và đường dẫn là bắt buộc để tính năng ghi đè hoạt động.

Hệ máy (System)	Core phổ biến	Đường dẫn Save mặc định (Virtual FS)	Phần mở rộng
NES	FCEUmm / Nestopia	/home/web_user/retroarch/userdata/saves/	.srm
SNES	Snes9x	/home/web_user/ret	.srm

		roarch/userdata/saves/	
GBA	mGBA / VBA-M	/home/web_user/retroarch/userdata/saves/	.srm / .sav
PlayStation	PCSX ReARMed	/home/web_user/retroarch/userdata/saves/	.srm / .mcr

Tên file save thường trùng với tên file ROM. Ví dụ: ROM là game.nes thì Save là game.srm.[4]

7.2. Quy trình "Tiêm" Save Game (Restore Process)

Thách thức lớn nhất là EmulatorJS có thể ghi đè file save trống ngay khi khởi động nếu không tìm thấy file tồn tại. Do đó, ta cần ghi file save từ Firebase vào Virtual FS *ngay khi* hệ thống tệp sẵn sàng nhưng *trước khi* game load file đó.

Giải pháp tối ưu là sử dụng hook EJS_onGameStart kết hợp với việc kiểm tra đối tượng Module.FS.

JavaScript

```
// Hàm chuyển đổi Base64 sang Uint8Array (Binary)
function base64ToUint8Array(base64) {
    const binaryString = window.atob(base64);
    const len = binaryString.length;
    const bytes = new Uint8Array(len);
    for (let i = 0; i < len; i++) {
        bytes[i] = binaryString.charCodeAt(i);
    }
    return bytes;
}

// Hook được gọi khi Emulator bắt đầu
window.EJS_onGameStart = async function() {
    console.log("Emulator Started. Checking cloud saves...");
```

```
    if (!auth.currentUser) return;

    const gameId = currentGameConfig.id; // ID từ gamelist.json
    const romName = currentGameConfig.rom_path.split('/').pop(); // Lấy tên file ROM
    const saveFileName = romName.replace(/\.\w+$/, '.srm'); // Đổi đuôi thành.srm

    // Tải dữ liệu từ Firebase
    const snapshot = await get(child(ref(db), `users/${auth.currentUser.uid}/saves/${gameId}`));

    if (snapshot.exists()) {
        const cloudData = snapshot.val();
        const saveBytes = base64ToUint8Array(cloudData.srm_data);

        // Đường dẫn trong Virtual FS
        const virtualPath = `/home/web_user/retroarch/userdata/saves/${saveFileName}`;

        try {
            // Kiểm tra xem FS đã sẵn sàng chưa và ghi file
            if (window.Module && window.Module.FS) {
                // Tạo thư mục cha nếu chưa tồn tại
                const dir = '/home/web_user/retroarch/userdata/saves';
                window.Module.FS.createPath('/home/web_user/retroarch/userdata', 'saves', true, true);

                // Ghi file save vào bộ nhớ
                window.Module.FS.writeFile(virtualPath, saveBytes);
                console.log(`Restored save to ${virtualPath}`);

                // Quan trọng: Yêu cầu Core load lại save nếu cần thiết
                // Trong một số trường hợp, cần restart core để nhận file mới
                // window.EJS_emulator.restart();
            }
        } catch (e) {
            console.error("Failed to inject save:", e);
        }
    }
};
```

Một số tài liệu gợi ý rằng việc ghi file trong EJS_onGameStart có thể trễ hơn thời điểm core đọc save. Trong trường hợp đó, ta cần một chiến lược "Soft Restart": Cho game chạy, ghi file save, sau đó tự động kích hoạt lệnh Reset giả lập để game load lại dữ liệu từ SRAM.[18]

7.3. Quy trình "Trích xuất" Save Game (Backup Process)

Để đồng bộ ngược lên Cloud, ta cần bắt sự kiện khi nội dung file save thay đổi. EmulatorJS cung cấp hook EJS_onSaveUpdate.

JavaScript

```
// Hàm chuyển đổi Uint8Array sang Base64
function uint8ArrayToBase64(bytes) {
    let binary = '';
    const len = bytes.byteLength;
    for (let i = 0; i < len; i++) {
        binary += String.fromCharCode(bytes[i]);
    }
    return window.btoa(binary);
}

window.EJS_onSaveUpdate = function() {
    console.log("Save update detected!");

    const romName = currentGameConfig.rom_path.split('/').pop();
    const saveFileName = romName.replace(/\.\w+$/, '.srm');
    const virtualPath = `/home/web_user/retroarch/userdata/saves/${saveFileName}`;

    try {
        if (window.Module && window.Module.FS) {
            // Đọc file từ Virtual FS
            const fileData = window.Module.FS.readFile(virtualPath);

            // Encode sang Base64
            const base64String = uint8ArrayToBase64(fileData);

            // Đẩy lên Firebase
            const updatePayload = {
                srm_data: base64String,
                timestamp: Date.now()
            };

            set(ref(db, `users/${auth.currentUser.uid}/saves/${currentGameConfig.id}`), updatePayload)
                .then(() => console.log("Synced to Cloud successfully"))
                .catch((err) => console.error("Sync failed", err));
        }
```

```
    } catch (e) {
        console.error("Error extracting save:", e);
    }
};
```

Ngoài hook EJS_onSaveUpdate, việc sử dụng setInterval để kiểm tra và backup định kỳ (ví dụ mỗi 60 giây) là một lớp bảo vệ bổ sung cần thiết, phòng trường hợp hook không được kích hoạt đúng lúc hoặc trình duyệt bị đóng đột ngột.[1]

---

8. Tối ưu hóa Hiệu năng và Xử lý Sự cố

Khi triển khai thực tế, một số vấn đề về hiệu năng và logic có thể phát sinh.

8.1. Tối ưu hóa Base64 và Băng thông

Việc mã hóa Base64 làm tăng kích thước dữ liệu lên khoảng 33%. Với các file save SRAM (thường < 128KB), điều này không đáng kể. Tuy nhiên, nếu áp dụng cho Save States (có thể lên tới vài MB), độ trễ sẽ tăng cao.

- **Giải pháp:** Chỉ đồng bộ file .srm (in-game save) qua Realtime Database. Nếu cần đồng bộ Save State, hãy sử dụng **Firebase Cloud Storage** và chỉ lưu URL tham chiếu trong Realtime Database.
- **Nén dữ liệu:** Có thể sử dụng thư viện LZ-String để nén chuỗi Base64 trước khi gửi lên Firebase, giúp giảm băng thông và dung lượng lưu trữ.

8.2. Xung đột Dữ liệu (Concurrency Control)

Vấn đề xảy ra khi người dùng chơi trên thiết bị A (offline), sau đó chơi trên thiết bị B, rồi thiết bị A online trở lại và ghi đè dữ liệu cũ lên thiết bị B.

- **Giải pháp:** Luôn kiểm tra timestamp. Trước khi ghi đè lên Cloud, so sánh local_timestamp và cloud_timestamp. Chỉ ghi nếu local > cloud. Ngược lại, nếu cloud > local khi khởi động game, hiển thị thông báo cho người dùng chọn phiên bản muốn giữ.

8.3. Xử lý Trình duyệt Safari và iOS

Safari trên iOS có các chính sách quản lý bộ nhớ và Service Worker rất nghiêm ngặt. Việc sử dụng coi-serviceworker có thể gặp trục trặc trên các phiên bản iOS cũ. Cần kiểm tra kỹ lưỡng (testing) trên môi trường này. Đôi khi, việc tắt SharedArrayBuffer (chấp nhận chạy đơn luồng, hiệu năng thấp hơn) là giải pháp bắt buộc để hỗ trợ iOS ổn định.

---

9. Kết luận và Hướng dẫn Thực thi

Việc xây dựng hệ thống EmulatorJS với đồng bộ Firebase trên GitHub Pages là một minh chứng cho sức mạnh của công nghệ Web hiện đại. Nó loại bỏ rào cản phần cứng, cho phép trải nghiệm game di sản mọi lúc mọi nơi.

Tóm tắt các bước thực thi cốt lõi:

1. **Repository Setup:** Tạo GitHub Repo, thiết lập cấu trúc thư mục, thêm coi-serviceworker.js.
2. **Emulator Setup:** Tải EmulatorJS core vào thư mục /data, cấu hình index.html với Service Worker.
3. **Database Design:** Thiết lập Firebase Project, cấu hình Authentication và Realtime Database Security Rules.
4. **Integration Logic:** Viết script main.js để xử lý logic:
   - Auth User.
   - Load ROM từ gamelist.json.
   - Inject Save (Cloud -> Virtual FS) tại EJS_onGameStart.
   - Extract Save (Virtual FS -> Cloud) tại EJS_onSaveUpdate.
5. **Testing:** Kiểm tra kỹ lưỡng quy trình Save/Load trên nhiều trình duyệt và thiết bị khác nhau.

Giải pháp này không chỉ giải quyết bài toán kỹ thuật về giả lập trên web mà còn mở ra hướng đi cho các ứng dụng web serverless phức tạp khác cần quản lý trạng thái người dùng bền vững.

Báo cáo được tổng hợp dựa trên các tài liệu kỹ thuật về EmulatorJS, RetroArch, Emscripten File System và Firebase SDK Documentation.

Nguồn trích dẫn

1. Options - EmulatorJS, truy cập vào tháng 1 14, 2026, https://emulatorjs.org/docs/options/
2. EmulatorJS - RomM, truy cập vào tháng 1 14, 2026, https://docs.romm.app/4.5.0/Platforms-and-Players/EmulatorJS-Player/
3. Save Data · EmulatorJS · Discussion #400 - GitHub, truy cập vào tháng 1 14, 2026, https://github.com/orgs/EmulatorJS/discussions/400
4. Local Saves · Issue #2 · linuxserver/docker-emulatorjs - GitHub, truy cập vào tháng 1 14, 2026, https://github.com/linuxserver/docker-emulatorjs/issues/2
5. Downloading save from Web Player? - RetroArch - Reddit, truy cập vào tháng 1 14, 2026, https://www.reddit.com/r/RetroArch/comments/mmvlt3/downloading_save_from_web_player/
6. Allow setting COOP and COEP headers in Github Pages · community · Discussion #13309, truy cập vào tháng 1 14, 2026, https://github.com/orgs/community/discussions/13309

7. truy cập vào tháng 1 14, 2026, https://docs.wasmer.io/sdk/wasmer-js/how-to/coop-coep-headers
8. Is there any way to use SharedArrayBuffer on GitHub Pages? - Stack Overflow, truy cập vào tháng 1 14, 2026, https://stackoverflow.com/questions/68609682/is-there-any-way-to-use-sharedarraybuffer-on-github-pages
9. node.js - How can I save a base64-encoded image to disk? - Stack Overflow, truy cập vào tháng 1 14, 2026, https://stackoverflow.com/questions/6926016/how-can-i-save-a-base64-encoded-image-to-disk
10. How to reconstruct a file from base64 encoding? - Stack Overflow, truy cập vào tháng 1 14, 2026, https://stackoverflow.com/questions/60270199/how-to-reconstruct-a-file-from-base64-encoding
11. Storing and retrieving a base64 encoded string in Firebase storage - Stack Overflow, truy cập vào tháng 1 14, 2026, https://stackoverflow.com/questions/64180914/storing-and-retrieving-a-base64-encoded-string-in-firebase-storage
12. gzuidhof/coi-serviceworker: Cross-origin isolation (COOP and COEP) through a service worker for situations in which you can't control the headers (e.g. GH pages) - GitHub, truy cập vào tháng 1 14, 2026, https://github.com/gzuidhof/coi-serviceworker
13. Setting the COOP and COEP headers on static hosting like GitHub Pages - Blogccasion, truy cập vào tháng 1 14, 2026, https://blog.tomayac.com/2025/03/08/setting-coop-coep-headers-on-static-hosting-like-github-pages/
14. WebReflection/mini-coi: A minimalistic version of coi-serviceworker. - GitHub, truy cập vào tháng 1 14, 2026, https://github.com/WebReflection/mini-coi
15. How to use a file upload input as a variable (HTML)? - Stack Overflow, truy cập vào tháng 1 14, 2026, https://stackoverflow.com/questions/79248918/how-to-use-a-file-upload-input-as-a-variable-html
16. How to properly selfhost EmulatorJS - Reddit, truy cập vào tháng 1 14, 2026, https://www.reddit.com/r/selfhosted/comments/18qt8eo/how_to_properly_selfhost_emulatorjs/
17. [FEAT] Separate Save files into Directories by System/Emulator · Issue #48 · linuxserver/docker-emulatorjs - GitHub, truy cập vào tháng 1 14, 2026, https://github.com/linuxserver/docker-emulatorjs/issues/48
18. romm/frontend/src/views/Player/EmulatorJS/Player.vue at master - GitHub, truy cập vào tháng 1 14, 2026, https://github.com/rommapp/romm/blob/master/frontend/src/views/Player/EmulatorJS/Player.vue