ChatGPT

# Giới thiệu tổng quan về Mozilla Hubs

Mozilla Hubs là một nền tảng không gian 3D đa người trên trình duyệt (WebVR/WebXR) mã nguồn mở do Mozilla phát triển. Hubs hoạt động như một ứng dụng web chạy trên trình duyệt, chứa mã HTML/CSS/JS để mô phỏng một "thế giới 3D mạng" và hiển thị giao diện người dùng 2D cho các menu và điều khiển [1] [2]. Hubs dùng công nghệ A-Frame (với Three.js) để dựng cảnh 3D và BitECS cho kiến trúc ECS (quản lý trạng thái game) [3]. Hệ thống Hubs bao gồm cả phần client và các dịch vụ backend: Reticulum (server Phoenix quản lý trạng thái và hiện diện mạng), Dialog (NodeJS/Janus SFU xử lý truyền âm thanh/ hình ảnh), cùng các thành phần khác như Networked-AFrame adapter và Janus Gateway [4] [5].

# Cấu trúc thư mục của mã nguồn Hubs

Mã nguồn chính của Hubs được chia thành các phần lớn (thư mục chính):
- **admin**: Ứng dụng điều khiển (admin panel) riêng biệt.
- **react-components**: Thư viện các component React dùng cho giao diện 2D (menu, cửa sổ, toolbar) của client.
- **src**: Toàn bộ mã logic của ứng dụng 3D, bao gồm hệ thống ECS, các component cho thực thể 3D, hệ thống mạng, v.v. (entry point và cấu hình được xác định trong `webpack.config.js` [6]).
- Ngoài ra còn có các thư mục phụ trợ như **scripts** (công cụ tiện ích), **test** (kiểm thử), **types** (định nghĩa TypeScript), cùng các file cấu hình (.env, tsconfig, package.json...) và file cảnh (scene) hoặc giao diện mẫu (ví dụ **habitat** lưu asset mặc định).

Trong thư mục `src`, mã được tổ chức thành các module chính như **components** (định nghĩa dữ liệu ECS và component), **systems** (hệ thống xử lý liên quan), **app** (khởi tạo thế giới và luồng chính), **utils** (hàm tiện ích cho mạng, JSON schema, v.v.), **scenes** (các scene 3D), **networking** (đồng bộ hóa trạng thái),... Ví dụ, file `src/utils/hub-channel.js` định nghĩa kênh Phoenix (HubChannel) dùng để đồng bộ dữ liệu giữa client và server [7].

# Kiến trúc hệ thống (client-server)

Hubs có kiến trúc client-server kết hợp hai loại giao tiếp chính: **WebSockets/Phoenix Channels** để truyền trạng thái trò chơi và **WebRTC/SFU (Selective Forwarding Unit)** để truyền dữ liệu đa phương tiện (âm thanh, video).

- **Client Web App**: Chạy trên trình duyệt của người dùng, xây dựng bằng A-Frame (Three.js) kết hợp BitECS. Client chịu trách nhiệm tải scene 3D (qua HTTP), tạo và cập nhật các thực thể (avatars, vật thể...), và gửi/nhận các cập nhật trạng thái qua mạng.
- **Reticulum (Phoenix Backend)**: Một hệ thống server Elixir/Phoenix theo kiến trúc *mesh* chịu trách nhiệm quản lý trạng thái hiện diện và chuyển tiếp dữ liệu thời gian thực. Mỗi client khi tham gia phòng (room) sẽ kết nối qua WebSocket đến một node Phoenix trên lưới Reticulum. Các thông điệp mạng từ client (cập nhật vị trí avatar, trạng thái component, v.v.) được gửi lên kênh Phoenix

(HubChannel), sau đó Reticulum truyền tiếp dưới dạng pub/sub đến các client khác trong phòng [8]. Phoenix Channels đảm bảo tường thuật các cập nhật trạng thái (được đóng gói thường bằng JSON) giữa các client một cách "eventual consistent" [9] [7]. Các thực thể *được ghim* (pinned entities) sẽ được lưu vào cơ sở dữ liệu Postgres của Reticulum để tồn tại sau khi client tạo ra nó rời phòng [10].

- **Janus/Dialog (SFU)**: Hệ thống đa phương tiện của Hubs dùng mô hình SFU. Client dùng WebRTC để gửi dữ liệu âm thanh/ảnh (từ microphone, webcam) lên một SFU trung tâm (cụ thể là node *Dialog* viết bằng NodeJS sử dụng Janus SFU) [11]. Janus nhận các luồng media và chuyển tiếp đến các client khác trong phòng. Mạng giữa client và Janus (và giữa Janus và Reticulum) được triển khai qua kết nối bảo mật wss, thường được cấu hình thông qua DialogAdapter trong mã nguồn client. Thông tin về phiên (session) WebRTC, SDP và ICE được xử lý nội bộ trong Dialog và adapter, tách biệt khỏi luồng dữ liệu của Reticulum [12] [11].
- **Networked A-Frame (NAF)**: Hubs sử dụng thư viện *Networked-AFrame* để đồng bộ hóa các thực thể A-Frame. Mỗi thực thể muốn đồng bộ được gắn component `networked` và được quản lý bởi một `networked-scene`. NAF gửi/nhận trạng thái component giữa các client thông qua adapter (ở đây là adapter của Hubs để dùng WebSocket Janus) [13] [14].

Tóm lại, luồng dữ liệu điển hình khi một client di chuyển avatar sẽ như sau: client ghi nhận vị trí mới, đóng gói dữ liệu thành message theo schema định nghĩa và gửi lên kênh `HubChannel` của Reticulum (qua WebSocket Phoenix). Reticulum nhận được, phát tín hiệu cập nhật này đến các client khác kết nối phòng, mỗi client nhận về cập nhật và áp dụng cho thực thể tương ứng. Trong khi đó, dữ liệu âm thanh/ video của avatar được gửi tới SFU Dialog (qua WebRTC) và chuyển tiếp đến tất cả client còn lại trong phòng [8] [4].

# Mô hình dữ liệu của Hubs

Mô hình dữ liệu của Hubs bao gồm nhiều thành phần: **cảnh 3D**, **thực thể (entities)** và **trạng thái components**.

- **Scene 3D**: Các phòng (rooms) trong Hubs được thiết kế bằng công cụ Spoke và xuất ra định dạng GLB (glTF). Client tải GLB này qua HTTP và nhúng vào A-Frame scene với Three.js quản lý. Cảnh 3D bao gồm địa hình, vật thể, ánh sáng,... và được tổ chức theo cây cảnh (scene graph) của Three.js. Dữ liệu GLB chứa mesh, texture, animation... cho avatar và môi trường.
- **Entity – Component (ECS)**: Bên cạnh cây cảnh, Hubs giữ trạng thái thế giới bằng hệ thống ECS của *bitECS* [15]. Mỗi **entity** là một đối tượng có ID độc nhất, và đi kèm một tập **component data** (ví dụ vị trí, góc quay, animation state, avatar customisation). Dữ liệu component được lưu trong các `TypedArray` tối ưu. Ví dụ, vị trí và góc xoay (Position, Rotation) được sync liên tục; thông tin biểu cảm, avatar attachment, trạng thái điều khiển (ví dụ đang nói) cũng có thể được lưu trong các component tương ứng. Khi cần đồng bộ, client chỉ gửi những component thuộc *schema mạng* đã định nghĩa. Các schema này (định nghĩa trong mã nguồn như trong `src/utils/network-schemas.js`) chỉ rõ các trường dữ liệu nào sẽ đồng bộ [7].
- **Avatar state**: Avatar của người dùng được tạo từ mô hình 3D cơ sở (robot template) có thể tùy chỉnh skin. Trạng thái avatar gồm kiểu mô hình, texture, và các component vị trí (vị trí thân, tay, chân, đầu). Hubs đồng bộ hóa các biến dạng đốt xương (joint) hoặc các trigger animation để khi người dùng vận động (chạy, nhảy, giơ tay) thì biểu diễn nhất quán trên các client. Dữ liệu này được biểu diễn dưới dạng giá trị số (float) trong component (ví dụ giá trị xoay khớp) và được gửi qua Phoenix dưới dạng JSON.
- **Định dạng dữ liệu mạng**: Các thông báo qua WebSocket chủ yếu là các payload JSON chứa thông tin entity ID và giá trị component đã cập nhật. Ví dụ một message có thể giống

`{"m":"update","e":123,"c":{"p":{"x":1.0,"y":2.0,"z":3.0},"r":{"x":0,"y":90,"z":0}}}`, trong đó `e` là entity ID, `c` chứa component `"p"` (position) và `"r"` (rotation). Dữ liệu WebRTC (âm thanh/video) không dùng JSON mà là các luồng nhị phân (MediaStream); tuy nhiên việc khởi tạo phiên WebRTC giữa client và Janus cũng sử dụng signaling (SDP) qua WebSocket.

- **Đồng bộ hóa**: Mạng đồng bộ theo cơ chế "eventual consistency": Reticulum chỉ biết gửi tín hiệu, còn mỗi client tự nhận và áp dụng dữ liệu theo cách của mình. Client thường gửi cập nhật định kỳ (ví dụ 10 lần/giây) chứ không mỗi khi trạng thái thay đổi ngay lập tức [16]. Khi client rời phòng, các entity do client đó tạo ra sẽ bị xóa trừ khi được đánh dấu *pinned* (lưu vào DB) [10].

# Các mô-đun quan trọng và luồng xử lý chính

- **Reticulum (Phoenix)**: Mô-đun trung tâm xử lý mạng lưới của Hubs. Mỗi node Reticulum chạy Phoenix và giữ một phần cơ sở dữ liệu chung (Postgres) để lưu người dùng, phòng, metadata. Khi client tham gia phòng, server sẽ gán cho nó kênh `HubChannel`. Tất cả thông điệp cập nhật vị trí, trạng thái của thực thể từ client sẽ được gửi qua kênh này [8]. Server Reticulum không hiểu nội dung cụ thể của thông điệp, chỉ phát tán đến mọi client khác trong phòng. HubChannel client-side (trong `src/utils/hub-channel.js`) thực hiện việc nối kênh và thiết lập sự kiện ngắt/đồng bộ (presence) [7]. Các entity được ghim sẽ được lưu ở server, cho phép tái tạo hiện trạng khi có client mới tham gia.
- **Networked-AFrame (NAF) và Janus Adapter**: Hubs dùng NAF để quản lý việc thêm/bớt thực thể trên scene và đồng bộ hóa component. Hệ thống này sử dụng adapter tuỳ chỉnh để kết nối NAF với SFU Janus qua WebSocket. Khi một thực thể mới "spawn" (ví dụ ai đó thêm một ảnh, video hoặc nhãn tên), NAF báo cho server và các client khác tạo thực thể tương ứng theo một `prefab`. Mã nguồn định nghĩa các prefab và schema (trong `src/utils/network-schemas.js`) cho từng loại thực thể. Ví dụ, prefab cho avatar sẽ bao gồm thông tin mô hình và cài đặt animations.
- **Dialog và Janus Gateway**: Dialog là thành phần NodeJS tích hợp Janus SFU để xử lý audio/video chat. Client Hubs sẽ khởi tạo kết nối WebRTC đến Dialog để truyền stream giọng nói và hình ảnh chân dung. Janus cung cấp một plugin (metadata SFU) cho phép tích hợp các stream này vào scene ảo (ví dụ hiển thị video của người dùng trên một plane trong phòng). Mô-đun *DialogAdapter* bên client chịu trách nhiệm config và kết nối đến Dialog qua WebSocket, rồi khởi tạo các peer connection cần thiết. Luồng xử lý: khi người dùng bật microphone, client gọi API của adapter để gửi âm thanh đến Dialog; Dialog sau đó gửi ngược lại audio của các user khác.
- **Xử lý Luồng chính (Game Loop)**: Client Hubs có một vòng lặp cập nhật (mainTick) để xử lý logic game. Các sự kiện mạng (từ Reticulum) được xếp hàng đợi và áp dụng tại mỗi tick. Hệ thống ECS cập nhật vị trí và hoạt hình của các entity, chạy physics (nếu có), xử lý va chạm, và phát âm thanh/hiển thị video tương ứng. Ví dụ, component "AvatarRig" có thể xử lý chuyển động khớp tay chân; component "Voice" có thể gán âm thanh thu từ WebRTC vào hướng của tai avatar. Các hệ thống này được viết trong thư mục `src/systems`.
- **Các dịch vụ phụ trợ khác**: *Hubs-Ops* chứa mã hạ tầng (Terraform, Docker) để triển khai các dịch vụ AWS; *Spoke* (github.com/hubsfoundation/spoke) là editor scene (tách biệt); *Asset server* lưu trữ file (hình, 3D)... Có thể đề cập qua ví dụ cấu trúc thư mục hoặc tài liệu, nhưng chính báo cáo tập trung vào code hệ thống Hubs-CE.

# Kết luận và gợi ý cải tiến

Mozilla Hubs là một hệ thống phức hợp kết hợp nhiều công nghệ (A-Frame/Three.js, ECS, WebRTC, Phoenix) để tạo ra trải nghiệm đa người trên trình duyệt. Cấu trúc mã nguồn tuy lớn nhưng có sự phân chia rõ ràng giữa thành phần UI React và phần logic 3D/ mạng. Để cải tiến, nhóm có thể xem xét:

- **Nâng cấp công nghệ**: Ví dụ cập nhật A-Frame/Three.js lên phiên bản mới, hoặc chuyển dần sang WebGPU. Xem xét dùng state management mới (như Redux hoặc các hooks React) cho phần UI nếu thấy phù hợp.
- **Tài liệu và minh hoạ**: Tạo sơ đồ luồng dữ liệu (như flowchart kết nối client–Reticulum–Janus) để dễ hiểu. Bổ sung các comment inline trong mã quan trọng (như `HubChannel`) để giúp người mới.
- **Tối ưu hóa đồng bộ**: Xem lại tần suất gửi dữ liệu và schema mạng để tránh quá tải. Cân nhắc dùng delta compression hoặc chỉ gửi khi có thay đổi lớn.
- **An toàn và bảo mật**: Kiểm tra các cập nhật TLS/SSL của WebSocket, xác thực client tốt hơn (hệ thống hiện dùng magic link email), và bảo mật dữ liệu người dùng (avatar, room).
- **Tính năng mở rộng**: Bổ sung WebXR Input (tay ảo), scripts để người dùng tự động hóa hành vi; hoặc tích hợp NFT/Web3 nếu mục tiêu dự án là "web3/metaverse".

**Tóm lại**, báo cáo nên đi từ tổng quan hệ thống và mã nguồn đến từng thành phần chi tiết (thư mục, các file quan trọng như `hub-channel.js`, các schema trong `utils/network-schemas.js`...), kết hợp ví dụ mã và trích dẫn cụ thể để bạn đọc dễ theo dõi. Cấu trúc đề xuất (Giới thiệu – Thư mục – Kiến trúc – Mô hình dữ liệu – Mô-đun/luồng – Kết luận) sẽ giúp nhóm triển khai nội dung một cách rõ ràng và có hệ thống.

**Nguồn tham khảo:** Tài liệu và mã nguồn Hubs (GitHub) [2] [5] [15], tài liệu phát triển Hubs (Hubs Foundation Docs) [5] [15], bài viết kỹ thuật về WebRTC trong Hubs [4]. Các đường dẫn cụ thể trong đó trỏ đến file mã và tài liệu chính thức để nhóm dễ tra cứu thêm.

---


[1] [3] [6] [15] Hubs Client development Basics · Hubs
https://docs.hubsfoundation.org/dev-client-basics.html

[2] GitHub - Hubs-Foundation/hubs: Duck-themed multi-user virtual spaces in WebVR. Built with A-Frame.
https://github.com/Hubs-Foundation/hubs

[4] [11] How Mozilla Hubs Uses WebRTC < Hubs WebRTC Tester
https://zachfox.io/hubs-webrtc-tester/about/

[5] [7] [8] [9] [10] [12] [16] Hubs Client development Networking · Hubs
https://docs.hubsfoundation.org/dev-client-networking.html

[13] Mozilla Hubs | hubs
https://webaverse.github.io/hubs/

[14] GitHub - networked-aframe/networked-aframe: A web framework for building multi-user virtual reality experiences.
https://github.com/networked-aframe/networked-aframe