ChatGPT

Spectra Decentralized Architecture & Collaborative Sandbox

Distributed Network Architecture and Deployment

Spectra Gallery's infrastructure is designed as a **distributed, decentralized network** spanning local servers and cloud nodes. The core intranet consists of a Fedora 42 server hosting the main services (using systemd and Podman containers), with additional Ubuntu Server nodes (including devices like Raspberry Pi running Ubuntu) joined via a WireGuard VPN mesh [1]. This forms a **private LAN/WAN pool** (a "constellation" of nodes) where services can discover each other via mDNS on the LAN and communicate securely over the VPN. The architecture extends to cloud providers as well – for example, Windows VMs on Azure (for specialized Xbox GDK workloads) and Linux VMs on Google Cloud – all **peered into the same WireGuard network**, creating a unified address space across on-premise and cloud [2] [3]. In essence, each node (whether a local server or cloud VM) becomes part of a **decentralized ecosystem** with layered network topology (local LAN, VPN overlay, cloud extension), enabling interoperability across the "interweb" of LAN and WAN resources.

This distributed design allows Spectra's stack to be **scalable and resilient**. Fedora acts as a central hub, while Ubuntu/Raspberry Pi nodes can host microservices or provide extra compute, all without exposing internal traffic to the public internet. Instead, **cross-site traffic is routed through the VPN**, keeping internal API endpoints private [1]. For instance, a Raspberry Pi on-site could run a service (e.g. a sensor data collector or rendering worker) and communicate with the Fedora server over the WireGuard tunnel. Likewise, cloud instances join the mesh and can be accessed as if they were on the local network. This **"holographic" network architecture** means any part of the system can interact with any other as peers in a secure overlay, rather than through a single centralized server.

Deployment of the stack across these nodes is managed through standard DevOps practices. Each service (frontend, backend, storage, playground, etc.) runs independently as its own process or container and exposes HTTP APIs [4] [5]. Continuous integration and deployment scripts (e.g. using GitHub Actions or simple bash scripts) pull the latest code to each server, install dependencies, and restart services [6]. This allows updates to propagate through the constellation of nodes smoothly. The **self-hosted server pool** can thus include heterogeneous environments (Fedora, Ubuntu, cloud VMs) all running the Spectra services in tandem. As long as a new node can join the WireGuard VPN (for which configuration is shared across the cluster) and meets basic requirements (Node.js 18, MongoDB, etc.), it can **host parts of the Spectra ecosystem**. In practice, even a Raspberry Pi could be configured as an Ubuntu node in the VPN to run lightweight tasks or local sensor interfaces, contributing to the overall system.

Interactive Collaborative Sandbox (Playground) Integration

At the heart of Spectra's front-end is an **interactive coding sandbox** called the **Playground**, which enables collaborative creative coding directly in the web interface. The Spectra Frontend (a Nuxt SSR app) includes a

component named `PlaygroundModal` dedicated to this "sandbox" functionality [7]. The PlaygroundModal provides a pop-up IDE with **multiple editor panels and a live preview**. Specifically, it contains three code editors – one each for HTML, JavaScript, and CSS – side by side, along with an output panel that renders the result in real time [8]. Under the hood, these editors are powered by Ace (brace) in the browser and are coordinated via Vuex state. As the user writes code, the changes are injected into an iframe for preview, allowing for instantaneous feedback. The snippet below from `PlaygroundModal.vue` shows the structure with three `<BlockEditor>` components and a `<BlockResult>` preview iframe:

[8]

This in-browser playground is **persisted and integrated** with the Spectra backend, meaning users can save their "sketches" (code projects) and load them later. Each sketch consists of the HTML/JS/CSS content and metadata. The Playground leverages the main backend's persistence logic and APIs so that saved sketches are stored in MongoDB and associated with the user's account [9]. Users can thus create a portfolio of generative art or creative code snippets and retrieve them from any device. The PlaygroundModal component on creation generates a unique hash ID for a new project and, if the user is logged in, loads their existing sketches from the database [10] [11]. It also auto-saves work-in-progress: as you edit, the content is debounced and saved via an API call in the background after a few seconds of inactivity [12] [13]. This ensures the sandbox is not only interactive but also retains work seamlessly.

Importantly, the Playground is designed with **collaboration in mind**. The interface is launched from the main navigation ("Playground" link in the header) and opens as a modal so that it can overlay whatever page the user is on [14] [15]. In a single-user scenario, this already provides a powerful creative tool; however, Spectra envisions multi-user collaboration in this sandbox. The codebase includes a Socket.io integration, hinting at real-time collaborative editing support. The frontend registers a Socket.io client (connecting to a dedicated service on `localhost:4000`) to enable live updates [16]. Although the current backend's Socket.io server logic is scaffolded (and was at one point experimenting with "quantum" data encryption over sockets) [17] [18], the presence of this plugin suggests that **real-time co-editing** is on the roadmap. With WebSockets in place, multiple users could join the same sandbox session, with each keystroke broadcasted to others. Implementing a CRDT or operational transform on the code editors would allow the Spectra Playground to become a truly **collaborative sandbox**, where artists and developers in different locations see each other's changes live.

In summary, the PlaygroundModal provides a **dynamic, live-editing environment** embedded in Spectra's web app. It follows the familiar model of tools like CodePen or JSFiddle but is tailored to Spectra's platform (including integration with user accounts and neural mapping tools). This sandbox lowers the barrier for creative experimentation – artists can prototype generative visuals or interactive pieces directly on the gallery platform, share them, and even collaborate in real time as the infrastructure evolves to support it.

Certificate Authority and YubiKey-Based Security

To secure this distributed ecosystem, Spectra employs a combination of an internal Certificate Authority mechanism and **YubiKey-backed authentication**. The goal is to issue and trust certificates across all domains/nodes in the architecture, establishing a chain of trust within the decentralized network, while also leveraging hardware keys (YubiKeys) for strong user authentication and encryption tasks.

On the server side, the Spectra backend acts as a sort of certificate authority for the internal services. At startup, it will **generate or load an RSA key pair** that can be used for signing credentials or tokens [19]. The private key is encrypted with a server secret and stored (and only decrypted in memory on startup), ensuring that even at rest the CA key is protected. This RSA key can serve as the **root of trust** for issuing certificates. For example, when new services or nodes (e.g., a new Ubuntu server or container) join the network, they could generate a CSR (Certificate Signing Request) that the Spectra backend signs using this key, producing a certificate that all other nodes will trust (since they share the CA). In practice, this means internal API calls over TLS can be secured without resorting to public CAs – every node trusts the Spectra CA. The backend's key initialization code shows that if no key exists, it creates a new 2048-bit RSA pair, saves the encrypted private key and a PEM of the public key (which could be distributed as the CA cert), and logs the event [20]. By managing its own PKI in this way, the system supports **secure inter-service communication** even in an air-gapped or intranet environment.

For user-facing security, Spectra integrates **WebAuthn via YubiKey** to augment traditional login flows. After a user logs in (for example, via an external identity provider like SwissID or via Spectra's own credentials), they are prompted to register a YubiKey as a second factor [21]. The backend's YubiKey service (using the FIDO2 library) generates a challenge and expected parameters, and the user taps their YubiKey to create a new credential. This credential's public key is stored in MongoDB and tied to the user (or in the current implementation, a global credential store) [22]. Subsequent authentications can require the YubiKey signature (WebAuthn assertion) to ensure it's really the user. The **identity flow** thus goes: OAuth/OIDC login → establish session → WebAuthn registration/verification with YubiKey → allow access to sensitive actions [23] [24].

Beyond just authentication, Spectra uses the YubiKey to protect API operations as well. One example is the **Spectra Storage** service (handling uploads/media) which can be configured to require a valid YubiKey signature on requests. Because the user's YubiKey public key is known to the system, the storage service can challenge the user to sign a request (or simply trust a JWT that was issued only after YubiKey verification). This means even if a user's session token were stolen, critical operations could demand proof of possession of the hardware key [22]. The Spectra backend's YubiKey service module encapsulates this logic, providing functions to generate WebAuthn registration options, verify attestation, and even **sign data with the RSA private key** for issuing tokens or certificates [25] [26]. Thus, the combination of an internal CA and WebAuthn yields a robust security framework: the **CA issues certificates** to trusted nodes/services (establishing an authority across the "decentralized architecture"), and **WebAuthn ensures user identity** and ties client actions to a physical key.

In practical terms, to "generate an API issuing certificate across the domains," one could expose an endpoint (accessible to admin or automated scripts on new nodes) where a CSR can be posted and a signed cert returned, similar to how Let's Encrypt ACME works but on a private authority. The signing operation would utilize the backend's loaded private key (protected by the YubiKey-derived secret). By using a YubiKey (which could also potentially store the CA key via PIV module, or at least guard the secret to decrypt it), the process gains an extra layer of physical security – only someone with the YubiKey and proper access can initiate certificate issuance. This approach mirrors the design in Spectra's backend: **keys are generated and encrypted with a secret, and critical cryptographic operations require that secret (and by extension the hardware token)** [19]. Overall, Spectra's security design ensures that both the **network trust (certs)** and **user trust (auth)** are anchored in strong cryptography, with the convenience of an integrated system for issuing and using certificates in a decentralized manner.

Advanced Creative Features: WebGL, Sensors, and AI Integration

One of the most exciting aspects of Spectra's sandbox and overall platform is its support for **advanced web technologies** to enable rich creative coding experiences. The environment is equipped to handle WebGL and even WebGPU for rendering graphics, can interface with device sensors and gamepads for interactivity, and integrates AI "agents" for generative or assistive features. All of these are accessible through the live collaborative editor (Playground) and related tools, making Spectra's sandbox a powerful playground for experimental art and tech.

WebGL/WebGPU Rendering: The Spectra sandbox supports high-performance graphics directly in the browser. Example demo scenes include a "wormhole" shader that leverages WebGPU (with fallback to WebGL) for cutting-edge GPU rendering [27]. Users can write custom WebGL shaders or use Three.js and see the results in the preview panel. In fact, the repository's sample content includes scenes like a neural network of celestial bodies rendered with WebGL, audio-reactive 3D shapes, and physics simulations using Three.js [28]. The Playground's live iframe preview is capable of running any such code – since it's essentially a mini webpage, the full Web APIs are available. This means a user could prototype an interactive 3D artwork in the Playground, using `<canvas>` and WebGL contexts or the emerging WebGPU API, and visualize it instantly. The "Wormhole" demo, for instance, allows the user to drag or use a gamepad thumbstick to twist a vortex shader in real-time [27], illustrating the kind of dynamic visuals the platform can handle. By supporting WebGPU, Spectra is also somewhat future-proofing the sandbox for next-generation graphics (shaders, compute on GPU, etc.), all within the browser.

Sensor and Device Integration: The platform is built with an eye toward interactive art that goes beyond keyboard and mouse. There is unified support for sensor inputs and gamepad controls via an **InputManager** module [29]. This means that code running in the Playground or in associated apps can read from device sensors (like accelerometer, gyroscope, GPS) or game controllers and use that data in creative ways. For example, an art piece could use the viewer's device orientation to influence a visual effect, or a gamepad could be used to navigate a 3D scene displayed in the Playground. Because Spectra's stack can span to IoT-like devices (recall that Raspberry Pis can join the network), one could even imagine hooking up external sensors feeding into the system. The front-end supports standard Web APIs for sensors and controllers, which the user's code can access. The InputManager abstraction suggests that Spectra aims to make it easy to handle multiple input modalities in a consistent way, empowering creators to build **immersive, sensor-driven experiences** in the browser.

AI Agents and Autocompletion: Spectra incorporates AI both as a subject of creative exploration and as a tool to assist users. In the sandbox's context, there are references to various **autonomous agents** – for instance, the Arquolab sandbox repository includes generative agents like *AlphaEvolveAgent*, *SigmaAlgeaSwarm*, and *QuantumJellyfishAgent* which simulate evolutionary art and flocking behavior [30]. These can be run on the backend (with endpoints to start/stop them) and visualized in the Playground, allowing users to inject algorithmic intelligence into their art. On the user-assistance side, the sandbox also features an **AI autocomplete/chat integration**. By setting an OpenAI API key, a user can unlock a `/chat` page in the sandbox app where a GPT-based assistant can help with code or answer questions [31]. This could be used for getting code suggestions (autocompleting code snippets, for example) or for interactive guidance as one codes. In the future, such an agent might be directly integrated into the Playground editor, offering real-time code completion or design suggestions as an "AI pair programmer." For now, the groundwork is there: the system can interface with OpenAI's models, and the collaborative editor concept means those suggestions could even be shared among collaborators. The presence of a "neural map"

feature (accessible via playground server API routes) also hints at AI-driven visualization or knowledge mapping tools to assist in the creative process [32].

To summarize these features: - **Graphics**: Full support for WebGL and experimental WebGPU for rendering cutting-edge visuals in-browser [28]. - **Devices**: Access to device sensors and gamepads, unified by an InputManager, enabling interactive works that respond to physical movement or external inputs [29]. - **Agents**: Built-in generative agents (e.g. evolutionary art algorithms, swarms) that can be controlled via API and integrated into projects [30]. - **AI Assistance**: Optional integration with AI language models (OpenAI GPT) to provide code autocompletion, chat-based help, or generative content when the user opts in [31].

All of these are delivered through a **dynamic live editor** – the Spectra Playground – which can potentially be used collaboratively. This aligns with Spectra's goal of a *"holographic" creative experience*: many layers (graphics, hardware, AI, multi-user) interacting in real time. By following the Spectra Playground model and these integrations, developers and artists can co-create in a rich sandbox that merges visual coding, real-world data, and AI. The result is a truly **next-generation creative coding platform** that is an integral part of the Spectra Gallery ecosystem, supporting everything from generative art exhibitions to live collaborative performances across a decentralized network.

Sources:

1. Spectra Distributed Gaming Environment – *Architecture and Components (Fedora 42, Ubuntu nodes, WireGuard mesh, Azure/GCP cloud integration)* [2] [1]
2. Spectra Distributed Gaming Environment – *Network Architecture (LAN mesh, VPN, unified address space)* [33] [3]
3. Spectra Intranet Deployment Docs – *Repository Components (Frontend with PlaygroundModal, Backend with auth, Storage with WebAuthn, Playground server)* [34]
4. `PlaygroundModal.vue` – Vue component defining the collaborative code editor modal with multiple BlockEditors (HTML/JS/CSS) and a BlockResult preview [8]
5. Spectra Playground Server README – *Lightweight Express playground with three ACE editors and shared persistence with main backend (sketch save/load)* [9]
6. `HeaderBar.vue` – Nav menu integration showing "Playground" link that triggers the sandbox modal in the UI [14] [15]
7. `socket.plugin.js` (Frontend) – Initialization of Socket.io client for real-time collaboration features (connecting to localhost:4000) [16]
8. Spectra Backend `server.js` – Socket.io server scaffold (commented out) indicating planned real-time capabilities [17]
9. Spectra Backend YubiKey Service – *client.yubikey.service.js* (Key generation, encryption with SESSION_SECRET, WebAuthn flows) [19] [20]
10. Spectra Docs – *Identity Providers Integration (OAuth login + WebAuthn YubiKey 2FA + storage enforcement)* [21] [22]
11. Arquolab Sandbox Models README – *Features* (Generative art templates, AI agents, sensor/gamepad InputManager) [30] [29]
12. Arquolab Sandbox Models README – *WebGL/WebGPU Demos* (Wormhole demo, multiverse, torus, etc. with WebGPU fallback and controls) [27]
13. Arquolab Sandbox Models README – *OpenAI Integration* (Enabling `/chat` with OPENAI_API_KEY for GPT-based assistance) [31]

1 2 3 4 5 6 7 33 34 intranet-gaming-environment.md

https://github.com/spectra-gallery/spectra-afrhone/blob/494c1e913ad78211b1da84377d32f4e238a2c05b/docs/intranet-gaming-environment.md

8 10 11 PlaygroundModal.vue

https://github.com/spectra-gallery/spectra-afrhone/blob/494c1e913ad78211b1da84377d32f4e238a2c05b/spectra-frontend/components/PlaygroundModal.vue

9 32 README.md

https://github.com/spectra-gallery/spectra-afrhone/blob/494c1e913ad78211b1da84377d32f4e238a2c05b/spectra-playground-server/README.md

12 13 BlockEditor.vue

https://github.com/spectra-gallery/spectra-afrhone/blob/494c1e913ad78211b1da84377d32f4e238a2c05b/spectra-frontend/components/editor/BlockEditor.vue

14 15 HeaderBar.vue

https://github.com/spectra-gallery/spectra-afrhone/blob/494c1e913ad78211b1da84377d32f4e238a2c05b/spectra-frontend/components/HeaderBar.vue

16 socket.plugin.js

https://github.com/spectra-gallery/spectra-afrhone/blob/494c1e913ad78211b1da84377d32f4e238a2c05b/spectra-frontend/plugins/socket.plugin.js

17 18 server.js

https://github.com/spectra-gallery/spectra-afrhone/blob/494c1e913ad78211b1da84377d32f4e238a2c05b/spectra-backend/server.js

19 20 25 26 client.yubikey.service.js

https://github.com/spectra-gallery/spectra-afrhone/blob/494c1e913ad78211b1da84377d32f4e238a2c05b/spectra-backend/services/client.yubikey.service.js

21 22 23 24 identity-providers.md

https://github.com/spectra-gallery/spectra-afrhone/blob/494c1e913ad78211b1da84377d32f4e238a2c05b/docs/identity-providers.md

27 28 29 30 31 README.md

https://github.com/spectra-gallery/arquolab-sandbox-models/blob/0a1860ba765b9483f06ef4f0089f054806ac1037/README.md