The design decisions for OpenAura were made after carefully evaluating several architectural alternatives to ensure scalability, performance, and developer flexibility. For real-time communication, we compared Peer-to-Peer (P2P) and Selective Forwarding Unit (SFU) architectures. While P2P offers low latency in small groups, it struggles with bandwidth and scalability as participant numbers grow. SFU, on the other hand, allows broadcasters to send a single stream that can be relayed to many viewers efficiently. Hence, OpenAura adopts an SFU model using Mediasoup, providing scalable and reliable media streaming for large audiences. For API design, REST was chosen over GraphQL due to its simplicity, broad adoption, and seamless integration with Express.js. Although GraphQL offers flexible and efficient querying, REST better supports our current static endpoints while keeping the learning curve low for contributors. Regarding data storage, OpenAura uses MongoDB, a NoSQL database, over traditional relational databases like MySQL. This choice supports dynamic and semi-structured data such as user profiles, session data, and event configurations, which benefit from MongoDB's schema flexibility and horizontal scalability. The selected tech stack includes Express.js for backend routing, Socket.IO for real-time communication, React for the frontend, and Tailwind CSS for responsive UI styling. Multer handles file uploads, Cookie-parser manages authentication cookies, and Docker ensures environment consistency. These decisions align with OpenAura's goals of delivering a low-latency, real-time platform that is scalable, customizable, and developer-friendly.

To ensure OpenAura remains competitive, scalable, and aligned with evolving user needs, several future improvements and strategic directions are recommended. Firstly, implementing adaptive bitrate streaming using simulcast and SVC (Scalable Video Coding) would significantly improve viewer experience across diverse network conditions. Integrating AI-powered features such as real-time transcription, sentiment analysis, and automated moderation could further enhance the platform's usability and accessibility. Support for hybrid events—where physical and virtual attendees can interact seamlessly—should also be explored. Additionally, migrating the authentication system to a more secure, token-based OAuth2 flow or integrating with identity providers like Auth0 could boost security and ease enterprise adoption. On the analytics side, deeper event insights powered by machine learning could offer event organizers valuable data on audience engagement, drop-off rates, and session popularity. The introduction of monetization tools such as ticketing, in-stream donations, or sponsorship overlays would expand the platform’s appeal to commercial event organizers. Furthermore, transitioning from a monolithic deployment to a microservices architecture, potentially orchestrated with Kubernetes, would improve maintainability and scalability as the platform grows. Finally, offering a plugin system or SDK for third-party extensions would encourage ecosystem growth and community contributions, ensuring that OpenAura evolves as a flexible, extensible platform tailored to a wide range of use cases.