MetaHive: Virtual Workspace

Team Code Warriors

Nafis Fuad Shahid
MD Saidur Rahman Sagor
Md Abdul Muqtadir

January 17, 2025

Contents

1. MetaHive

MetaHive is a sophisticated virtual office platform designed to redefine remote work through an immersive 2D virtual environment. By integrating real-time interactions, AI-driven productivity tools, and a microservices-based backend architecture, MetaHive fosters a collaborative workspace that balances flexibility with accountability. This documentation provides an in-depth analysis of the frontend and backend architecture, features, and the rationale behind the solution's design.

2. Frontend Technical Documentation

The frontend of MetaHive crafts an engaging, user-centered interface that powers the immersive virtual office experience. Built with a combination of technologies like Next.js and Kaboom.js, it enables dynamic interactions, seamless communication, and efficient workflows.

2.1 2D Map and Avatar Navigation

Why It Matters: Introducing spatial dynamics into a remote work environment bridges the gap between physical and virtual offices. This approach not only enhances immersion but also replicates informal interactions crucial for team bonding, which traditional platforms often overlook.

The interactive 2D map allows users to navigate a virtual office space as avatars. This setup replicates the spatial dynamics of a physical office, fostering a sense of proximity among users.

Proximity-based triggers initiate features like video calls and screen sharing. For instance, as avatars approach one another, `AgoraRTC` seamlessly activates communication tools, creating an organic flow of interaction. `Kaboom.js` powers the game mechanics, ensuring responsive movement and interaction.

2.2 Document Collaboration

MetaHive integrates a document management system in one of its sidebars that mimics Confluence, enabling hierarchical organization of work materials. The interface includes advanced tools like text formatting, grammar checks, and AI-assisted summarization to streamline collaboration.

The AI-driven tools are designed to support users in creating high-quality outputs with minimal effort, reducing cognitive load and enhancing productivity.

2.3 Proximity-Based Calls and Screen Sharing

Why It Matters: Automating communication initiation based on proximity fosters spontaneity, which is a hallmark of effective in-person collaboration. This innovation eliminates manual interaction barriers, promoting natural workflows.

Calls and screen sharing are triggered automatically based on avatar proximity. This approach emulates spontaneous in-office discussions, reducing barriers to collaboration.

The synchronization between `Kaboom.js` for spatial logic and `AgoraRTC` for communication exemplifies the seamless interaction of frontend components.

2.4 Proctoring Interface

The proctoring feature offers real-time accountability by displaying snapshots captured by the backend's AI-driven monitoring system. Admins can review these logs to ensure compliance with organizational policies.

2.5 Real-Time Collaboration

Integrated tools like Discord provide additional communication channels for informal discussions, complementing the platform's structured features. `WebSocket` technology ensures low-latency synchronization of user actions and updates.

3. Backend Technical Documentation

The backend of MetaHive employs a robust Spring Boot microservices architecture, ensuring scalability, flexibility, and performance. It orchestrates critical functionalities such as user authentication, document storage, real-time interactions, and AI-driven features. Additionally, the system integrates a User Service to manage user roles and permissions seamlessly, enhancing the overall cohesiveness of the platform.

3.1 User Service

Manages user profiles, roles, and permissions, ensuring seamless authentication and access control throughout the platform. By centralizing user data, this service integrates smoothly with other microservices, particularly the Office Service and API Gateway, to maintain consistency and security.

Why It Matters: By handling complex role-based access control (RBAC) in collaboration with Keycloak, the User Service eliminates potential bottlenecks in user management, enabling a streamlined and secure user experience.

3.2 API Gateway

As the central point for request routing, the API Gateway ensures secure and efficient communication between clients and backend services. By integrating with Keycloak for token validation, it maintains consistent authentication practices while reducing redundancy across services.

3.3 Proctoring Service

Why It Matters: Ensures accountability in a distributed workforce without being overly intrusive. By intelligently identifying anomalies and securing evidence, it balances trust with oversight in a novel way that surpasses conventional monitoring techniques.

This service leverages `OpenCV` to detect user activity anomalies and log snapshots for administrative review. Its design prioritizes efficiency, ensuring accurate monitoring with minimal computational overhead.

3.4 Document Service

Facilitates structured document storage and collaboration with features like hierarchical organization. Integrated AI tools further enhance productivity by automating tasks like summarization and tone adjustments. The Document Service communicates with the Office Service to ensure that documents are uniquely tied to their respective offices, maintaining logical consistency and isolation across teams.

3.5 Game Server

Why It Matters: Acts as the real-time engine behind user interactions, enabling the seamless spatial logic necessary for the platform's unique features. Its contribution ensures a fluid, engaging, and responsive virtual workspace.

The game server synchronizes user interactions within the virtual map using `WebSocket` connections, ensuring real-time updates for features like proximity-based calls. It serves as the backbone for maintaining the dynamic and responsive nature of the 2D environment.

3.6 Office Service

Handles the creation and management of virtual offices, including team structures, role assignments, and permissions. This component ensures flexibility to accommodate diverse organizational needs.

4. System Flow

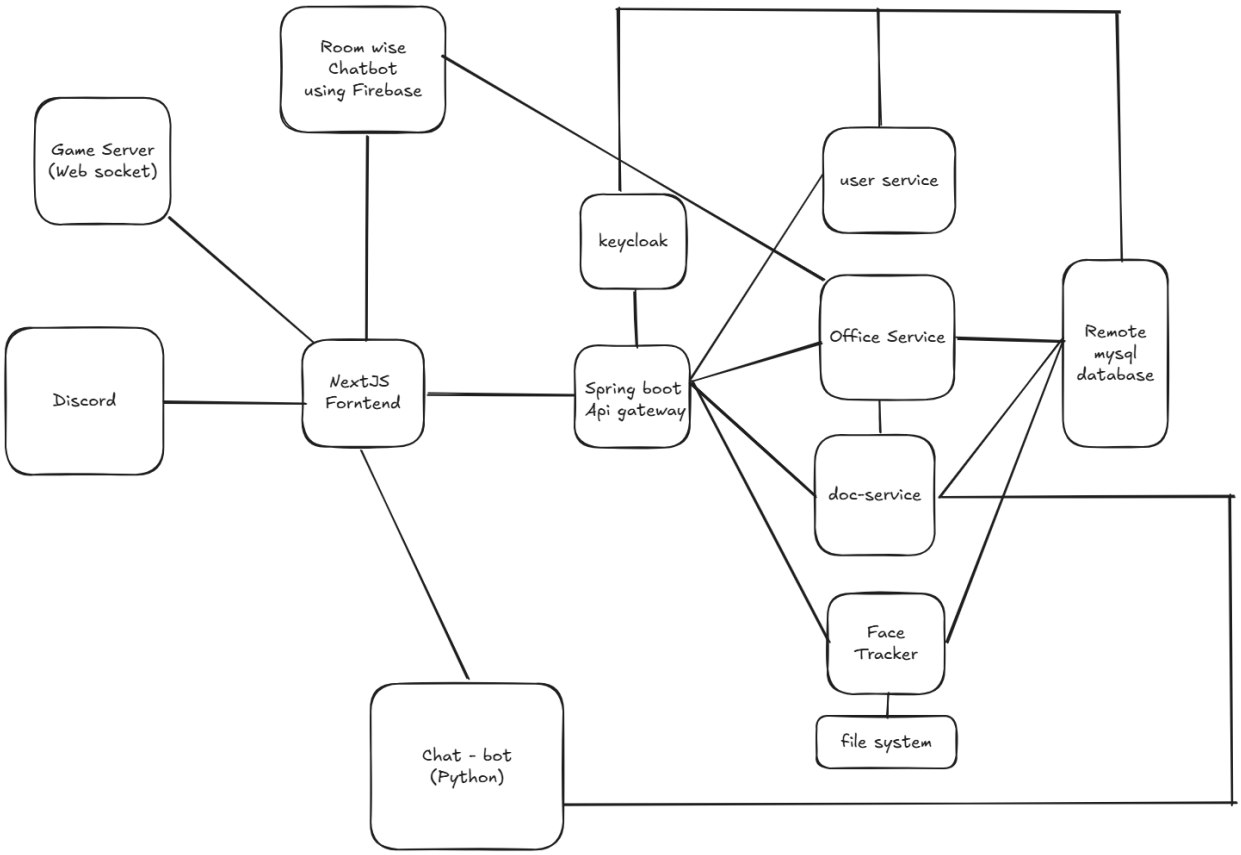

Figure 1: Backend Architecture

The MetaHive system architecture is structured to ensure secure, efficient, and scalable operations across its components. At the frontend, a **Next.js application** serves as the user interface, communicating with the backend through a centralized **Spring Boot API Gateway**. The API Gateway acts as a middleware, routing requests to appropriate microservices and handling user

authentication using **Keycloak**, which operates with **OAuth2**. During authentication, the frontend retrieves an access token from Keycloak and stores it in its cache. This token is subsequently included in all requests to the API Gateway, where it is validated before granting access to backend services.

The backend is built on a **microservices architecture**, with each service handling a specific domain. The **User Service** manages user profiles, roles, and permissions. The **Office Service** handles office-related data and operations, while the **Document Service** supports document editing and synchronization in real time. All microservices store their data in a **remote MySQL database**, ensuring consistency and persistence. In addition, the **Face Tracker** service monitors user presence and activity, primarily for proctoring and engagement tracking purposes.

Real-time interactions are supported through several key components. The **Game Server**, utilizing **WebSocket** technology, enables real-time synchronization of user actions and interactions within the virtual environment. A **room-wise chatbot**, built with **Firebase**, facilitates group-specific communication, while a Python-based chatbot provides enhanced conversational and support capabilities. Integration with external platforms like **Discord** further extends communication and collaboration beyond the core system.

To handle performance and scalability, the system employs **load balancing** and caching mechanisms, ensuring smooth operation under varying workloads. Each microservice is designed to operate independently, maintaining a modular and easily maintainable structure. Overall, the architecture balances **security**, **performance**, and **functionality**, providing a clear and efficient foundation for MetaHive's virtual office system.