SWED Excercise 2

* **Interfaces to 3rd Party Services:**
  + Managed primarily via the **Integration Service** and **Personio Marketplace**.
  + Uses **Public APIs** (e.g., Recruiting API, APIs for Attendances, Time off, Documents) for custom and partner integrations. API credentials control data access (read/write permissions per data type).
  + **Webhooks** can be configured for event-driven communication from Personio to third-party systems
  + Specific integrations include:
    - Payroll Systems (e.g., DATEV, Xero) for transferring salary-relevant data.
    - Job Boards (e.g., LinkedIn, StepStone) for posting vacancies.
    - Calendar Systems (e.g., Google Calendar, Outlook Calendar) for interview scheduling and absence synchronization.
    - Communication Tools (e.g., Slack) for notifications.
    - Identity Providers for SSO (e.g., Azure AD, Okta)
    - eSignature Providers (if not fully native)
    - Other business tools via the 200+ Marketplace integrations

**Describe your architecture and categorise your work within existing architectural styles.**

**Description:**  
The proposed architecture is a distributed system composed of multiple fine-grained, independent services, each focused on a specific HR domain (e.g., recruiting, time tracking, payroll prep). These services communicate over a network, typically via lightweight protocols like REST APIs orchestrated through an API Gateway. Each service manages its own data store, promoting loose coupling and independent scalability. The platform is designed as a multi-tenant SaaS application running in the cloud, leveraging cloud infrastructure for deployment, scaling, and reliability. Key aspects include strong integration capabilities via a dedicated service and marketplace, workflow automation driven by business events, and a focus on data analytics. Security features like role-based access and SSO are managed centrally but enforced across services.

**Categorization:**

* **Microservices Architecture:** This is the primary style, characterized by the decomposition of the application into small, autonomous services centered around business capabilities.
* **Cloud-Native Architecture:** Designed for and deployed on cloud infrastructure, leveraging patterns like containerization, dynamic orchestration, and managed services.
* **Service-Oriented Architecture (SOA):** Shares principles with SOA but typically emphasizes finer granularity and more decentralized governance than traditional SOA.
* **Event-Driven Architecture (Elements):** Workflow automation[3](https://www.personio.com/product/) and webhook capabilities[4](https://support.personio.de/hc/en-us/articles/4404623630993-Generate-and-manage-API-credentials) suggest the use of event-driven patterns for communication and process orchestration between services (e.g., an "employee hired" event triggering onboarding workflows).
* **Layered Architecture (Internal):** Individual microservices likely employ internal layering (e.g., API/Presentation, Business Logic, Data Access) for code organization

**2. Read the paper Yahyavi.pdf pages 1-20. Describe this architectural style and compare it to the styles mentioned in the lecture.**

**Description of the Peer-to-Peer (P2P) Architectural Style for MMOGs**

The architectural style described in the first 20 pages of the Yahyavi and Kemme paper2 is **Peer-to-Peer (P2P) Architecture applied to Massively Multiplayer Online Games (MMOGs)**. This style serves as an alternative to the traditional Client-Server model dominant in the MMOG industry2.

* **Core Idea:** Instead of relying entirely on powerful central servers or server farms, P2P architectures distribute the computational load, network bandwidth requirements, and sometimes game state storage among the participating player machines (peers)2. Peers connect and interact with each other directly or indirectly through a network overlay2.
* **Motivation:** The primary drivers for using P2P in MMOGs are:
  + **Scalability:** To support a massive number of simultaneous players by leveraging the collective resources of the peers2.
  + **Cost Reduction:** To lower the significant infrastructure and maintenance costs associated with large server farms required by client-server MMOGs2.
  + **Performance:** Potentially achieve faster response times (lower latency) through direct connections between interacting peers, especially for players in close proximity within the game world2.
* **Key Challenges:** P2P architectures for MMOGs face significant challenges:
  + **Control Complexity:** Managing a distributed game state consistently and fairly across many unreliable peers is complex2.
  + **Churn:** Peers frequently joining and leaving the network can destabilize the overlay and game state2.
  + **Cheating:** P2P systems are inherently more vulnerable to various forms of cheating, as peers might control parts of the game state or communication pathways2.
  + **NAT/Firewalls:** Establishing direct connections between peers behind home network address translators (NATs) and firewalls can be difficult2.
  + **Heterogeneity:** Peers have varying computational power, bandwidth (especially asymmetric upload/download speeds), and reliability2.
* **Fundamental Mechanisms:**
  + **State Distribution & Replication:** Game state (e.g., positions and attributes of avatars, objects) is often distributed. Many P2P MMOGs use a primary copy replication model where the authoritative copy of an object resides on one peer (or server in hybrid models), and other interested peers hold replicas2.
  + **Update Dissemination:** Changes in game state (events like movement, actions) are propagated from the primary copy holder to peers holding replicas. This often uses direct unicast messages to relevant peers or multicast mechanisms built on the P2P overlay (like Scribe)2.
  + **Interest Management (IM):** This is crucial for scalability. It limits the amount of game state information and updates a peer receives to only what is relevant to them, typically based on their position and vision range (Area of Interest - AOI) in the game world. Zoning (partitioning the game world) is a common IM technique2.
  + **Discovery:** Peers need mechanisms to find other relevant peers or the primary copies of objects within their AOI2.
  + **Consistency Control:** Ensuring all players have a sufficiently consistent view of the game world despite network latency and potential concurrency issues. Eventual consistency is often acceptable, and techniques like dead reckoning are used to mask latency2.
* **Main Variants Discussed (Pages 15-20):**
  + **Structured P2P:** Peers are organized into a specific, deterministic network topology (overlay), often using Distributed Hash Tables (DHTs) like Pastry or Chord. This structure facilitates efficient object lookup and message routing (e.g., finding the peer responsible for a game region or object). Examples include SimMud and Colyseus2.
  + **Unstructured P2P:** Peer connections are formed more randomly or based on proximity/interest, without a strict global structure. Discovery often relies on mechanisms like mutual notification among neighbors within the AOI or gossip protocols. Examples include pSense and all-to-all systems like Donnybrook2.
  + **Hybrid P2P:** These combine elements of P2P with traditional servers or mix structured and unstructured P2P approaches. Servers might handle critical tasks like authentication, persistence, or cheat detection, while peers handle gameplay communication. Examples include FreeMMG, MOPAR, and VoroGame2.

**Okay, let's compare the Peer-to-Peer (P2P) architectural style for Massively Multiplayer Online Games (MMOGs), as described in pages 1-20 of the Yahyavi.pdf2, with the styles listed in your screenshot1.**

* **P2P vs. Dataflow-centric:**
  + *Dataflow:* Focuses on the flow and transformation of data through a sequence of processing steps (like pipes and filters). The architecture is defined by the path data takes.
  + *Comparison:* P2P for MMOGs is not dataflow-centric. While game updates (data) flow between peers, the architecture is primarily concerned with managing distributed *state*, peer *interactions*, consistency, and discovery in a dynamic network, rather than a linear processing pipeline for data transformation2.
* **P2P vs. Data-centric:**
  + *Data-centric:* Revolves around a central, shared data store (repository) that components access and manipulate. Data integrity and accessibility via the central store are key.
  + *Comparison:* P2P for MMOGs is fundamentally different from a data-centric approach. It actively avoids centralization by distributing the game state (e.g., primary copies of objects) across multiple peers or servers2. Communication focuses on peer interactions and update propagation, not shared access to a single central repository. The goal is scalability through distribution, contrasting with the centralized nature of data-centric styles.
* **P2P vs. Hierarchical:**
  + *Hierarchical:* Organizes components in levels or layers with clear superior/subordinate relationships, often involving control flowing up or down (like a traditional layered architecture or a tree structure).
  + *Comparison:* P2P is not inherently hierarchical. While specific mechanisms within P2P systems can be hierarchical (e.g., multicast trees like Scribe2), the basic peer relationship is often flat or based on complex, non-hierarchical graph structures (DHT overlays, neighbor graphs in unstructured P2P)2. Peers are generally considered equals, coordinating amongst themselves rather than operating in strict layers of control.
* **P2P vs. Distributed Systems:**
  + *Distributed Systems:* A very broad classification for any system where components are located on different networked computers and communicate by passing messages.
  + *Comparison:* P2P architecture is a specific style within the general category of Distributed Systems2. It is characterized by its decentralization, direct peer interaction, resource sharing, and peer autonomy, distinguishing it from other distributed styles like Client-Server.
* **P2P vs. Event-Based:**
  + *Event-Based:* System components react to the occurrence of events. Communication is often asynchronous, using mechanisms like publish/subscribe. Components are loosely coupled and interact by producing and consuming events.
  + *Comparison:* P2P for MMOGs is highly event-based. Player actions, movements, and state changes are events that trigger updates2. The update dissemination mechanism, often using publish/subscribe via multicast (like Scribe) or direct notification based on AOI, is a core part of the architecture2. Peers react to incoming update events to maintain their local view of the game state. P2P heavily relies on event-driven communication for coordination and state synchronization.
* **P2P vs. Service-Oriented:**
  + *Service-Oriented Architecture (SOA):* Structures applications as a collection of discrete, discoverable services with well-defined interfaces (APIs) that communicate over a network, emphasizing loose coupling and reusability.
  + *Comparison:* While peers in a P2P system could abstractly be seen as providing "services" (like hosting object state), the P2P MMOG style differs significantly from typical SOA. P2P interactions are often more dynamic, transient, and based on game-specific logic (like AOI proximity) rather than formal, stable service contracts2. Discovery relies on P2P overlay mechanisms (DHT lookups, neighbor discovery) rather than service registries2. Communication protocols might be specialized for game traffic, not necessarily standardized web service protocols. The goals (scalability, low peer-peer latency) are also more specific than the general integration and reusability aims of SOA.

Ein Bild, das Text, Screenshot, Schrift, Zahl enthält.

KI-generierte Inhalte können fehlerhaft sein.

Okay, here are suggested architectural patterns for the specified software systems, along with justifications based on the search results and general architectural principles.

**4. Suggest an architectural pattern for the following software systems. Why would you recommend this pattern here?**

* **a) A whistleblowing system on the internet.**
  + **Recommended Pattern:** **Message Queue Architecture** combined with a **Layered Architecture**, potentially using **Event-Based** principles for processing.
  + **Why:**
    - **Anonymity & Decoupling:** The core requirement is whistleblower anonymity. A message queue strongly decouples the submission frontend from the backend processing. The whistleblower submits an encrypted report (possibly via Tor or a secure web form) to a queue, without direct interaction with the case handlers. This obscures the source and enhances resilience, as the message persists if the backend is temporarily unavailable.
    - **Security & Confidentiality:** A layered approach separates concerns: 1) Secure, anonymous submission interface, 2) Secure message queue, 3) Backend processing/case management layer accessible only to authorized personnel, 4) Secure, audit-proof data storage. Encryption should be applied end-to-end and at rest.
    - **Reliability & Integrity:** The queue ensures messages are not lost. Defined procedures and audit trails (revision security) are crucial for handling reports conscientiously.
    - **Flexibility:** Multiple reporting channels (web, hotline) can feed into the same secure processing pipeline via the queue or dedicated interfaces.
* **b) A video conferencing system.**
  + **Recommended Pattern:** **Hybrid Pattern: Client-Server + Peer-to-Peer (P2P) + SFU (Selective Forwarding Unit) / MCU (Multipoint Control Unit)**. Often leverages **Event-Based** signaling and **WebRTC** protocol15.
  + **Why:**
    - **Signaling & Control:** A Client-Server component is essential for managing sessions (signaling), authentication, user management, and coordinating connections (e.g., using SDP via a signaling server).
    - **Low Latency (Small Groups):** P2P connections can be used for 1-on-1 or very small calls to minimize latency by sending media streams directly between clients.
    - **Scalability (Group Calls):** For larger groups, P2P (Mesh) becomes inefficient due to high bandwidth and CPU load on clients. An SFU or MCU architecture is needed:
      * **SFU:** Receives one incoming stream from each participant and forwards multiple streams to other participants without transcoding. This reduces client upload requirements and server CPU load compared to MCU, but increases client download bandwidth and CPU load for processing multiple incoming streams. It's scalable and common.
      * **MCU:** Mixes all incoming streams on the server into a single composite stream tailored for each participant. This simplifies client requirements (only one incoming/outgoing stream) making it suitable for low-power devices, but requires significant server CPU power and is less scalable and more expensive than SFU.
    - **Distributed Architecture:** Using geographically distributed servers (SFUs/MCUs, potentially coordinated by a media request broker) reduces latency for users worldwide.
    - **WebRTC:** Provides the underlying protocols and APIs for real-time communication (media streaming, data channels) and handling network complexities like NAT traversal (using ICE).
* **c) A GPS tracker for cats.**
  + **Recommended Pattern:** **Client-Server Architecture** utilizing **IoT Protocols (e.g., MQTT)** and potentially a **Publish/Subscribe** pattern for data ingestion.
  + **Why:**
    - **Device-to-Cloud:** The fundamental interaction is the tracker device (client) sending location data to a central backend (server). A user application (web/mobile) acts as another client retrieving processed data from the server.
    - **Low Power & Efficiency:** Cat trackers are battery-powered and need to be lightweight. IoT protocols like MQTT are designed for constrained devices, minimizing power consumption and data overhead compared to protocols like HTTP.
    - **Scalability & Decoupling:** Using MQTT with a **Publish/Subscribe** pattern allows many devices to publish location data to specific topics (e.g., /location/<cat-id>) on an MQTT broker (part of the server backend). Backend services can subscribe to these topics to receive, process, and store the data (e.g., in a time-series database) without being directly coupled to the individual devices. This handles intermittent connectivity well and scales easily.
    - **Two-Way Communication (Optional):** MQTT also supports sending commands back to the device if needed (e.g., change reporting frequency, trigger radio beacon).
    - **Component Structure:** The architecture typically includes the device unit (GPS module, microcontroller, communication module like GSM/LTE-M/NB-IoT) and the server unit (data ingestion/broker, data processing, database, user application API)[1](https://aumraj.com/system-architecture-of-gps-vehicle-tracking-system-and-its-applications/).