Exercise 2

Sonntag, 27. April 2025 13:33

1)

Personio is a **SaaS company** offering **HR software** for **small and medium-sized enterprises (SMEs)**.

Its main functionalities include:

- **Applicant Tracking System (ATS):** Manage recruitment and onboarding
- **Employee Data Management:** Centralize employee information and documents
- **Time and Attendance Tracking:** Track attendance, time-off requests, and generate reports
- **Performance Management:** Support goal setting, feedback, and evaluations
- **Payroll and Benefits Management:** Automate payroll, taxes, and benefits
- **Learning and Development:** Manage training programs and track learning progress
- **HR Analytics and Reporting:** Analyze HR data through customizable dashboards

Structural Arhitecture:

1. **User Interface:**
   - Web Application
2. **Application Layer**
   - Employee Management Service
   - Payroll Service
   - Recruitment Service
   - Time an dAttendance Service
   - Reporting and Analytics Service
3. **Integration Layer:**
   - API Gateway
   - Authentication and Authorization Service
   - Notification Service (E-Mail, SMS)
4. **Data Layer:**
   - Relational Database (e.g. PostgreSQL)
   - NoSQL Database (e.g. MongDB)
   - Data Warehouse (for analytics and reporting)
5. **Third-Party Services:**
   - Tax Calculation Service
   - Benefits Management Service
   - External Job Boards
   - Payroll Providers
   - Cloud Storage Services (e.g. AWS S3)
6. **Infrastructure:**
   - Cloud Hosting (s.g. AWS, Azure)
   - CDN (Content Delivery Network)
   - Monitoring and Logging

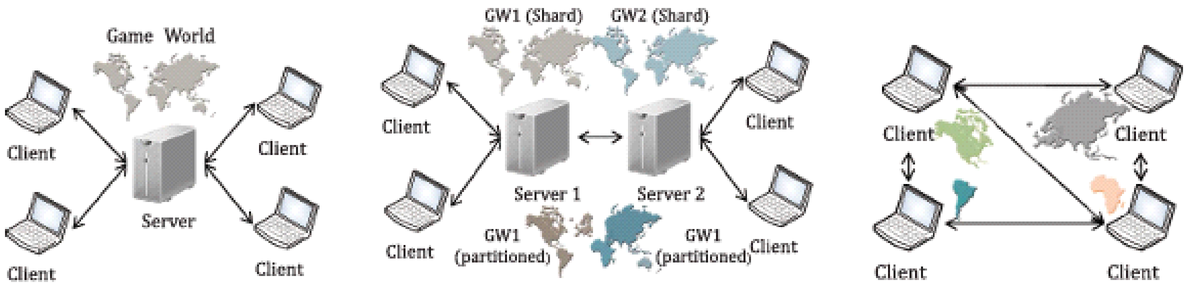
Architecture Description and Categorization:

- **Components**: The architecture is composed of modular services that handle different HR functions

- **Interfaces**: The API Gateway manages communication between the application and third-party services

- **Architectural Style**: Each HR function is encapsulated in its own service, allowing independent development, deployment, and scaling

2)



(a) Client-Server (b) Multi-Server (c) P2P

- **Client-Server Architecture**:
  - **Centralized control:** In client-server architecture, the server manages all game state, player actions, and conflict resolution.
  - **Client role:** Clients connect to the server to receive game data and send player actions.
  - **Scalability issue:** A single server has limited capacity; adding multiple servers is the usual solution.
- **Distributed Multi-Server Architecture**:
  - **Two types of multiserver architectures:**
    - **Shards**: Each server runs a full, separate game instance for its own group of players (no cross-server interaction).
    - **Single world with regions**: One shared game world is split into regions, each managed by a different server.
  - **Region-based architecture allows movement between servers**, often using a hand-off mechanism (automatic or via portals), enabling load balancing and interest management.
  - **Players are assigned to servers** (or regions) typically based on geography to reduce latency and distribute the load.
- **Peer-to-Peer (P2P) Architecture**:
  - **Peer-to-peer architecture** allows each node to act as both client and server.
  - **Scalability improves** because game data and updates are distributed across all nodes.
  - **Adding more players** increases available resources, reducing the load on any single node.

Characteristics and Comparison:

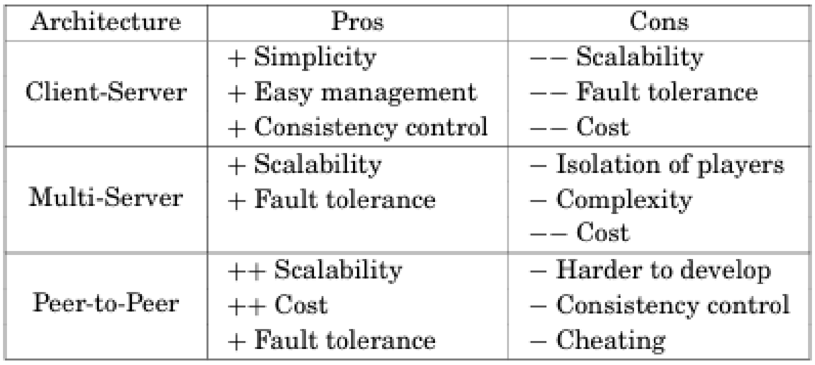
- **Centralized (server-based) architectures:** strong control, easy management, and simpler consistency handling
  - making them popular for persistent game worlds
- **Simplicity and ease of development**
  - major reasons game companies favor client-server models over more complex architectures like P2P
- **Main drawbacks**: poor scalability, high infrastructure costs, and being a **single point of failure** (which affects reliability and fault tolerance)
- **Backup servers:** can reduce risk, but they add cost and complexity, and may further limit

scalability

- **Multiserver architectures:** improve scalability and fault tolerance by dividing the game world into regions or shards, each managed by a separate server
- **Drawbacks:** limited player interaction across shards, complex region hand-offs, and overload issues in highly populated regions.
- **Scalability is limited**, as the world cannot be divided endlessly, and load balancing between regions is technically challenging
- **High costs** for server infrastructure, bandwidth, and maintenance make this architecture expensive (especially for smaller companies)

- **P2P architectures:** offer the highest scalability and lowest cost, as each player adds resources and reduces the need for central servers
- **Fault tolerance and latency** are improved through direct peer connections and distributed responsibility
- **Main drawbacks:** security risks (easier cheating), lack of centralized control, and increased complexity in managing consistency and coordination

Table I. Comparison of Different Architectures



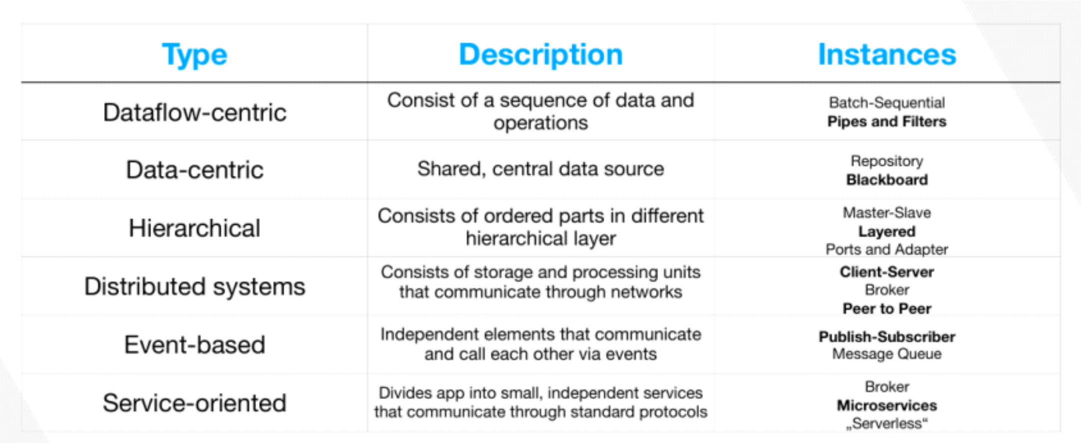
Architecture	Pros	Cons
Client-Server	+ Simplicity + Easy management + Consistency control	−− Scalability −− Fault tolerance −− Cost
Multi-Server	+ Scalability + Fault tolerance	− Isolation of players − Complexity −− Cost
Peer-to-Peer	++ Scalability ++ Cost + Fault tolerance	− Harder to develop − Consistency control − Cheating

- The paper discusses a **Peer-To-Peer Architecture**, characterized by:
  - **Decentralization**: No central server; each peer acts as both client and server
  - **Scalability**: New nodes can be added easily.
  - **Fault tolerance**: The system can handle failures of individual peers without affecting the overall network. – The System continues to work even if nodes fail
  - **Self-organization**: Nodes connect dynamically.
  - **Types**:
    - Unstructured (e.g., Gnutella) – random connections.
    - Structured (e.g., Chord) – efficient data search using algorithms (DHT).

Comparison with Other Architectures:

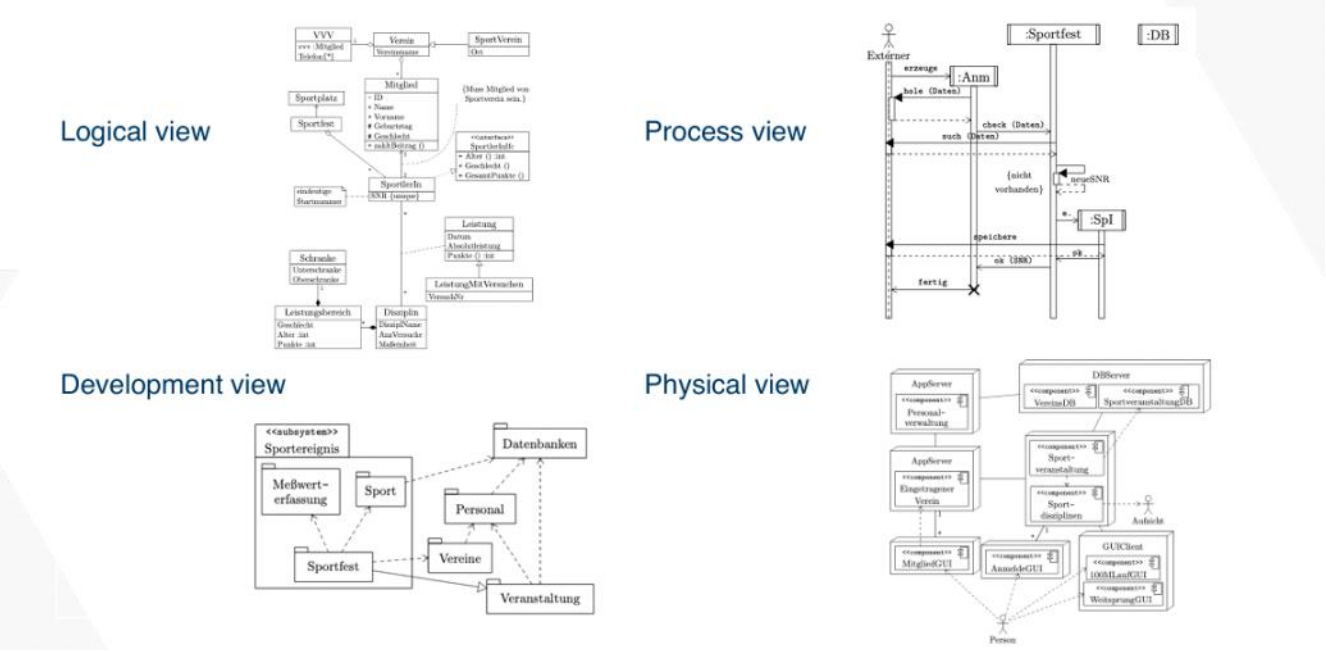
- **Client-Server Architecture**: Centralized with distinct roles for clients and servers, whereas P2P is decentralized
- **Microservices Architecture**: Composed of loosely coupled services focusing on specific business capabilities, whereas P2P focuses on distributing tasks among peers

- **Monolithic Architecture**: Single unified codebase, whereas P2P is distributed with each peer handling tasks independently



Type	Description	Instances
Dataflow-centric	Consist of a sequence of data and operations	Batch-Sequential **Pipes and Filters**
Data-centric	Shared, central data source	Repository **Blackboard**
Hierarchical	Consists of ordered parts in different hierarchical layer	Master-Slave **Layered** Ports and Adapter
Distributed systems	Consists of storage and processing units that communicate through networks	**Client-Server** Broker **Peer to Peer**
Event-based	Independent elements that communicate and call each other via events	**Publish-Subscriber** Message Queue
Service-oriented	Divides app into small, independent services that communicate through standard protocols	Broker **Microservices** „Serverless“

3)



Logical view

Process view

Development view

Physical view

- **Logical view**: shows the key abstractions in the system as objects or object classes
  - Class diagram
- **Process view**: shows how, at run-time, the system is composed of interacting processes
  - Sequence diagram
- **Development view**: shows how the software is decomposed for development
  - Package diagram
- **Physical view**: shows the system hardware and how software components are distributed across the processors in the system.
  - Deployment diagram
- (+1 is the use case view as outside perspective from the user)

Anmerkung Buono:

- Input und Output Type bei class diagram operations
- Klassen immer mit der richtigen Syntax benennen (keine Leerzeichen)
- Sequenz Diagramm: Klassen die erst von anderen erzeugt werden, müssen auch so

eingezeichnet werden (nicht einfach alle parallel oben nebeneinander)

4)

a) **Whistleblowing System on the Internet**: Microservices or Publisher/Subrciber architecture

   Microservices is a good choice for scalability and flexibility, while Publisher/Subscriber can be used for event-driven communication between users and administrators in real time.

   - they remain modular and adaptable
   - can react to new events in real time
   - and are independently scalable
   - handles **sensitive data and user anonymity** by isolating responsibilities

b) **Video Conferecning System**: Client Server architecture

   Client-Server architecture works well for video conferencing, as it enables centralized management of communication, media streaming, and user connections.

c) **GPS Tracker for Cats**: Client Server architecture or Publisher/Subscriber

   Client-Server is appropriate as the GPS tracker sends data to a central server, allowing real-time tracking and data management.