**1.1 Problem Statement & Requirements**

We need to design an **Authorization Microservice** that will manage **user authentication, roles, permissions, and access control** for multiple applications within a company.

**Key Functional Requirements**

1. **Multi-Application Support:**
   * The service must be accessible by multiple applications such as **web, mobile, and admin panel**.
   * Each application should have **customizable authentication and authorization policies**.
2. **User Authentication:**
   * The service should authenticate users via **OAuth 2.0, OpenID Connect (OIDC), or SAML**.
   * Support **multiple authentication providers** (e.g., **LDAP, Google, Facebook, GitHub**).
   * Implement **password hashing** using **bcrypt or Argon2**.
3. **Role-based Access Control (RBAC):**
   * Define **roles (Admin, User, Guest, etc.)** and assign **permissions (read, write, delete)**.
   * Store user-role mappings in a **database** for easy management.
   * Support **hierarchical roles** for better flexibility.
4. **Token-based Authorization:**
   * Use **JWT (JSON Web Tokens)** for stateless authentication.
   * Implement **short-lived access tokens** and **refresh tokens**.
   * Provide APIs to **generate, validate, and revoke tokens**.
5. **Auditing and Logging:**
   * Log **all authentication requests, permission checks, and access attempts**.
   * Store logs in **Elasticsearch, Loki, or a relational database** for analysis.
   * Enable **real-time monitoring & alerts** for suspicious activities.
6. **Scalability:**
   * Use **microservices architecture** to handle **millions of requests**.
   * Deploy using **Docker & Kubernetes** for **auto-scaling**.
   * Implement **Redis caching** for **fast token validation**.
   * Use **event-driven mechanisms** (e.g., Kafka, RabbitMQ) to handle updates asynchronously.

**High-Level Workflow**

1. **User Authentication:**
   * The user logs in via **OAuth 2.0 / OpenID Connect (OIDC) / SAML**.
   * The system validates credentials and **issues a JWT token**.
2. **Authorization Check:**
   * A user requests access to a resource.
   * The system validates the JWT token and checks **RBAC rules**.
   * If the user has the correct role and permissions, access is granted.
3. **Logging & Auditing:**
   * Every authentication and authorization event is **logged**.
   * Admins can track **who accessed what and when**.

### ****1.2 Key Design Considerations****

When designing an **Authorization Microservice**, we need to focus on security, scalability, and extensibility:

#### ****Security Considerations:****

1. **Authentication Protocols:** Use secure protocols like **OAuth 2.0, OpenID Connect (OIDC),** and **JWT (JSON Web Token)** for authentication.
2. **Encryption & Hashing:** Store passwords securely using **bcrypt, Argon2, or PBKDF2**. Encrypt sensitive data at rest and in transit using **TLS**.
3. **Token Security:** Use **short-lived access tokens** and **refresh tokens** to minimize security risks. Implement **token revocation** and **blacklisting** for compromised sessions.
4. **Principle of Least Privilege (PoLP):** Users should only have the permissions necessary to perform their actions.
5. **API Security:** Secure endpoints using **API Gateway**, implement **rate limiting**, and use **mTLS (mutual TLS)** for inter-service communication.
6. **Audit Logging:** Maintain detailed logs for authentication attempts, permission checks, and user actions for auditing and debugging.

#### ****Scalability Considerations:****

1. **Microservices Architecture:** Deploy the Authorization Service as a **standalone microservice**, allowing it to handle authentication/authorization independently.
2. **Horizontal Scaling:** Use **containerized deployment** with **Docker + Kubernetes** to scale dynamically based on load.
3. **Distributed Cache:** Implement **Redis** or **Memcached** for session caching and quick token validation.
4. **Event-Driven Architecture:** Use **Kafka** or **RabbitMQ** for handling role/permission changes asynchronously.

#### ****Extensibility Considerations:****

1. **Multi-Tenancy Support:** Allow multiple applications to use the service with **customizable roles/permissions**.
2. **Policy-based Access Control (PBAC):** Extend beyond RBAC to support **dynamic policy-based** authorization using **Open Policy Agent (OPA)**.
3. **Pluggable Authentication Providers:** Support multiple authentication mechanisms (**LDAP, SAML, OAuth, Social Logins**).
4. **Fine-Grained Access Control:** Define access at different levels: **User level, Role level, Resource level**.
5. **API Gateway Integration:** Use **Kong, Apigee, or AWS API Gateway** to centralize security policies.

### ****1.3 Technologies & Protocols****

#### ****Authentication & Authorization:****

* **OAuth 2.0 + OpenID Connect (OIDC)** – Standard for authentication and authorization.
* **JWT (JSON Web Token)** – Used for token-based authorization.
* **LDAP / SAML / Social Authentication (Google, Facebook, etc.)** – Optional authentication methods.

#### ****Database & Storage:****

* **PostgreSQL / MySQL** – For storing user roles, permissions, and access control lists (ACLs).
* **Redis / Memcached** – For caching user sessions and tokens.
* **Elasticsearch / Loki** – For logging and monitoring access events.

#### ****Infrastructure & Scalability:****

* **Docker & Kubernetes (K8s)** – For containerized deployment.
* **API Gateway (Kong / AWS API Gateway / Apigee)** – For securing APIs.
* **Kafka / RabbitMQ** – For event-driven access control updates.
* **Prometheus + Grafana** – For monitoring system performance.

#### ****Security & Compliance:****

* **mTLS (mutual TLS) & TLS 1.3** – For secure communication.
* **Keycloak / Auth0 / AWS Cognito** – For managing authentication flows.
* **Hashicorp Vault** – For managing secrets and API keys.
* **SIEM (Security Information and Event Management)** – For auditing and compliance.

## ****2. Microservice Architecture****

### ****2.1 Multi-Application Support & Communication Flow****

* The service will handle authentication and authorization for **web, mobile, and admin panel applications**.
* Each application will authenticate users via **OAuth 2.0 / OpenID Connect**, obtaining **JWT tokens** to interact with protected resources.
* The microservice will expose **RESTful API endpoints** for role and permission management.

### ****2.2 Multi-Tenancy****

* Each application can have **separate roles and permissions** stored in a multi-tenant-aware database.
* Applications will have unique client IDs and secret keys to authenticate with the microservice.

### ****2.3 API Endpoints****

#### ****Authentication APIs****

* POST /auth/login – Authenticate user and issue a JWT token
* POST /auth/refresh – Generate a new access token using a refresh token
* POST /auth/logout – Invalidate the current session

#### ****Role & Permission Management****

* POST /roles – Create a new role
* GET /roles – Retrieve all roles
* POST /permissions – Assign permissions to roles
* GET /permissions/:role – Retrieve permissions for a role

#### ****Access Control APIs****

* GET /access/resource – Validate if a user has access to a specific resource
* GET /audit/logs – Retrieve access logs for compliance and debugging

## ****3. Security & Authentication****

### ****3.1 Authentication Approach****

* Implement **OAuth 2.0 + OpenID Connect**.
* Use **JWT (JSON Web Tokens)** for stateless authentication.
* Support **LDAP and social logins** (Google, Facebook, GitHub, etc.).

### ****3.2 Token Management****

* **Access Tokens**: Short-lived JWT tokens (~15 minutes expiration)
* **Refresh Tokens**: Long-lived tokens (~7 days), securely stored, and used for re-authentication
* Secure token transmission via **HTTPS with TLS 1.3**

### ****3.3 Security Measures****

* **SQL Injection Protection**: Use ORM with parameterized queries
* **Cross-Site Scripting (XSS)**: Sanitize user inputs and use HTTP security headers
* **Cross-Site Request Forgery (CSRF)**: Implement anti-CSRF tokens for stateful interactions

## ****4. Data Model & Database Design****

### ****4.1 Entities & Relationships****

#### ****Entities:****

* **Users** (user\_id, username, email, password\_hash, is\_active)
* **Roles** (role\_id, role\_name, description)
* **Permissions** (permission\_id, permission\_name, description)
* **User\_Roles** (user\_id, role\_id) – Many-to-many mapping
* **Role\_Permissions** (role\_id, permission\_id) – Many-to-many mapping
* **Audit\_Logs** (log\_id, user\_id, action, timestamp, resource, status)

### ****4.2 Storage Strategies****

* Use **PostgreSQL / MySQL** for structured data.
* **Redis / Memcached** for caching frequently accessed permissions.
* **Elasticsearch / Loki** for storing audit logs.

## ****5. Scalability & High Availability****

### ****5.1 Scaling Strategy****

* **Microservices Deployment:** Each service instance runs independently.
* **Horizontal Scaling:** Deploy multiple instances using **Docker + Kubernetes**.
* **Load Balancing:** Use **NGINX / API Gateway** to distribute traffic.
* **Distributed Caching:** Use **Redis** to cache authentication tokens and permissions.

### ****5.2 Fault Tolerance & Availability****

* Implement **circuit breakers** (e.g., Netflix Hystrix) to prevent cascading failures.
* Use **replicated databases** with **failover mechanisms**.
* Enable **auto-recovery and self-healing** using Kubernetes.

## ****6. Rate Limiting & Security****

### ****6.1 API Protection Mechanisms****

* **Rate Limiting:** Restrict API calls per user/IP using **Redis-based throttling**.
* **IP Blocking:** Block repeated failed login attempts (brute force prevention).
* **API Key Management:** Require applications to authenticate API requests.

### ****6.2 Sensitive Data Protection****

* Use **environment variables** to store sensitive credentials.
* Avoid exposing user credentials in logs or API responses.
* Encrypt sensitive fields in the database using **AES encryption**.

## ****7. Auditing & Logging****

### ****7.1 Logging Strategy****

* Log all **authentication attempts, role changes, and permission checks**.
* Store logs in **Elasticsearch / Loki** with **structured JSON format**.

### ****7.2 Audit Log Data****

* User ID
* Timestamp
* Resource accessed
* Action performed
* Authorization status (granted/denied)
* IP address & user agent

### ****7.3 Compliance with Security Standards****

* **GDPR Compliance:** Allow users to request deletion of personal data.
* **SOC 2 / ISO 27001 Compliance:** Maintain audit logs for regulatory requirements.

## ****8. Conclusion****

This Authorization Microservice provides **secure authentication, fine-grained access control, scalability, and auditing**. By leveraging **OAuth 2.0, JWTs, and microservices architecture**, it ensures high security and performance across multiple applications.

