**System Design Document (SDD)**

## Project Name: GoCloak Prepared by: Sajib Jahan Date: March 16, 2025

|  |  |  |  |
| --- | --- | --- | --- |
| **Date** | **version** | **Prepared by** | **Checked by** |
| 20/03/2025 | 1.0.0 | Sajib Jahan | Masihur Maruf |

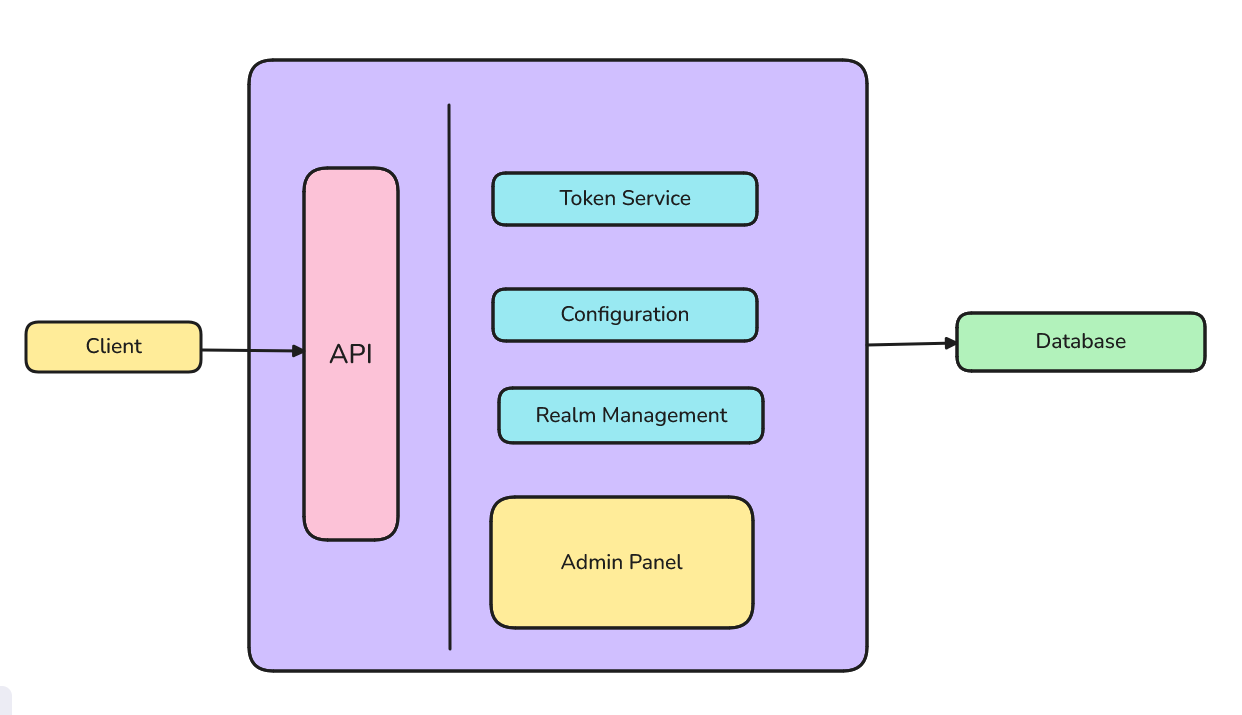
### **1. Overview**

**Purpose:** This System Design Document describes the architecture, component design, data flow, and security considerations for GoCloak—a lightweight, high-performance identity and access management (IAM) system written in Go. It aims to be a self-hosted alternative to Keycloak with modular REST APIs and an internal Admin Panel.

**Scope:** This document covers the following components:

* Admin Panel
* REST API Service
* Realm & User Management
* Token Service (OAuth2/OIDC)
* Role & Permission Management
* Database Design
* Security and Scalability Considerations
* Load Balancer
* Authentication and Authorization Modules
* Caching
* Federation Module
* External Sources
* Session Management

### **2. System Architecture Diagram**

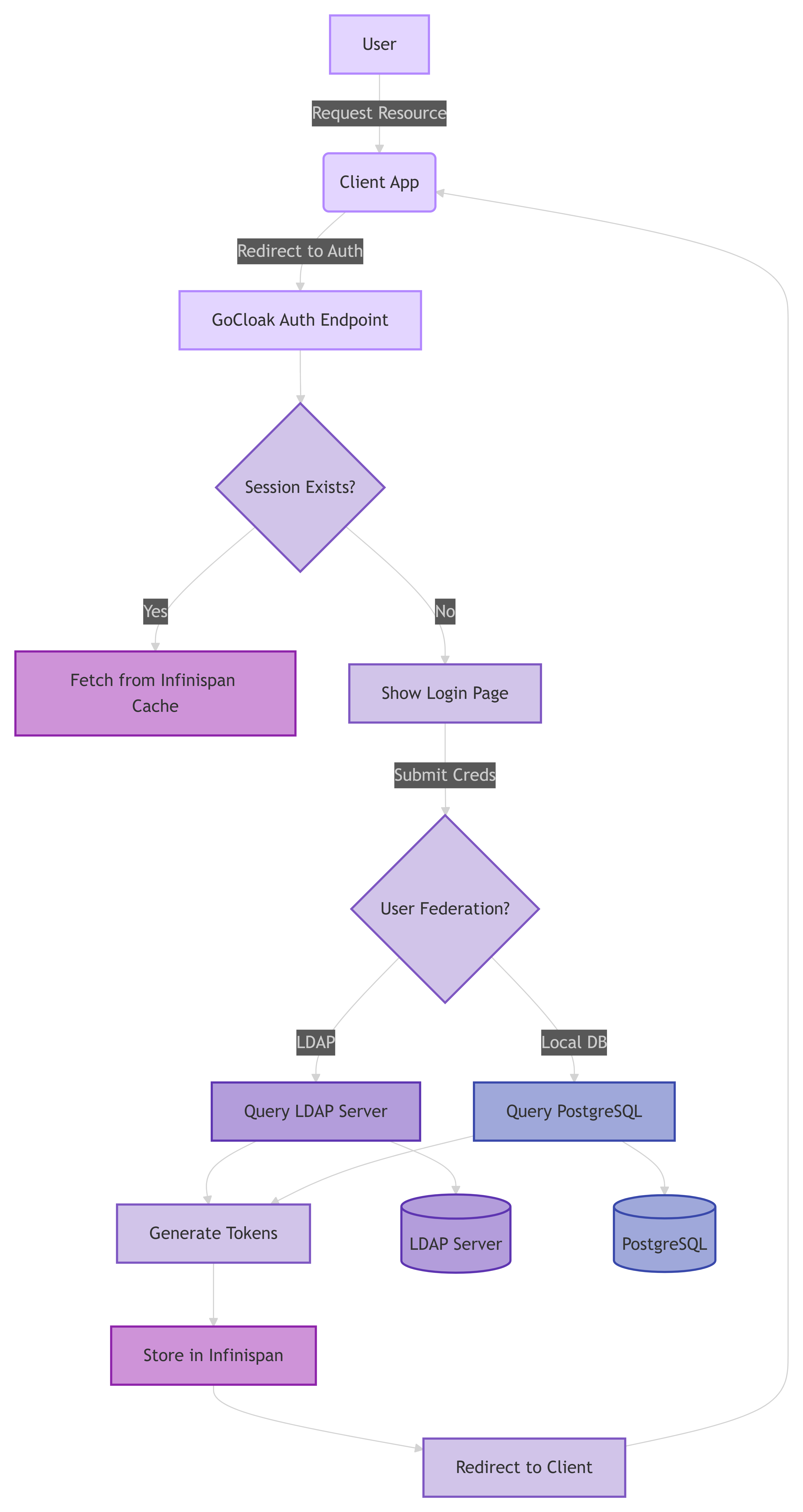


**Client:** Any backend machine, public frontend, mobile or device requesting IAM service

**API:** API endpoints that allows clients to communicate with GoCloak

**Admin Panel:** Web based view directly built in GoCloak core for managing realms & configurations

**Token service:** Service that facilitates token generation



# Technical Requirements

#### Platform

* **Programming Language**: Go (Golang) for performance, concurrency, and scalability.
* **Web Framework**: **Gin** or **Echo** for building the REST APIs.
* **Database**: **PostgreSQL** for production; **SQLite** for development.
* **Token Format**: **JWT (JSON Web Tokens)** for OAuth2 and OIDC support.
* **Authentication Protocols**: **OAuth2**, **OIDC**.
* **Deployment**: Dockerized for easy deployment; Kubernetes-compatible via Helm chart.
* **Security**: Enforce **HTTPS**, **CSRF protection** for the Admin Panel, and **JWT validation** for API access.

#### Security

* Use **JWT** for token management (RS256 or HS256).
* Ensure **HTTPS** for all public-facing endpoints.
* Implement **CSRF protection** in Admin Panel and input validation throughout.

### **3. Component Breakdown**

#### **3.1 REST API Service**

* Handles external communication via REST.
* Provides endpoints for managing realms, users, roles, and clients.
* Implements OAuth2/OIDC token issuance.
* Built using Go with the Gin framework.

#### **3.2 Admin Panel**

* Internal UI for administrators.
* Uses server-side rendering or HTMX/Tailwind.
* Interacts directly with the internal API layer.
* Restricted via admin login with role-based access control.

#### **3.3 Realm Manager**

* Manages logical tenant boundaries.
* Each realm is isolated: its users, roles, clients, and settings.
* Realm configurations stored in DB.

#### **3.4 User & Role Manager**

* Handles CRUD operations for users, groups, and roles.
* Supports password reset, role assignment, and status flags.

#### **3.5 Token Service**

* Issues access tokens (JWT - RS256).
* Refresh token and introspection support.
* Validates client credentials and user login.

### **4. Data Flow Diagrams**

#### **4.1 User Login Flow**

1. User submits credentials to /realms/{realm}/login.
2. Auth service validates and fetches users from DB.
3. Token service issues JWT.
4. Response returns access & refresh token.

#### **4.2 Client Credential Flow**

1. Client sends credentials to the token endpoint.
2. Service verifies client secret.
3. Token is issued for the client.

### 

### **5. Database Design**

#### **5.1 Tables**

* **Realms**: id, name, config\_json
* **Users**: id, realm\_id, username, password\_hash, status
* **Clients**: id, realm\_id, name, secret, redirect\_uris
* **Roles**: id, realm\_id, name, description
* **UserRoles**: user\_id, role\_id
* **AuditLogs**: action, performed\_by, entity\_type, timestamp

**Database:** PostgreSQL (SQLite for development mode)

### **6. API Gateway & Routing Strategy**

* /realms/{realm}/... scoped endpoints
* Versioning via headers or URI
* Throttling and rate-limiting optional in API gateway
* JWT-based bearer token validation

### **7. Security Considerations**

* HTTPS enforcement (TLS)
* Passwords hashed with bcrypt or argon2
* JWT signed with RS256
* Admin panel CSRF and XSS protection
* Input validation and sanitization
* RBAC for Admin Panel and API scopes

### **8. Scalability & Performance**

* Stateless services for easy scaling
* Containerized deployment (Docker)
* Load-balanced token services
* Configurable DB connection pooling
* Expected to support 100K concurrent sessions per realm

### **9. Extensibility & Modularity**

* Provider model for future integrations (social logins, LDAP)
* Plugin-based token pipeline (e.g., for audit logging)
* Configurable realms via JSON or admin UI

### **10. Deployment Architecture**

* Docker Compose for local
* Optional Helm Chart for Kubernetes
* CI/CD with GitHub Actions
* Reverse proxy (Caddy/Nginx) for routing and TLS termination

### **11. Monitoring & Logging**

* Audit logs stored per realm
* Metrics endpoint for Prometheus
* Grafana dashboards for user/session analytics
* Alerting on token service failures or DB outages

**10. Risks & Mitigations**

|  |  |
| --- | --- |
| High initial development complexity | MVP will limit features to essentials |
| Admin UI becoming overly complex | Focus on UX and modular components |
| Security vulnerabilities | Penetration testing and audit reviews |
| Scalability under load | Load testing and stateless architecture |

### **11. Appendices**

* API Reference - will be added
* Sequence diagrams (login, token flow) - will be added
* DB Schema diagrams - will be added