# **Technical Specification Document (TSD)**

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

|  |  |  |  |
| --- | --- | --- | --- |
| **Date** | **version** | **Prepared by** | **Checked by** |
| 16/03/2025 | 1.0.0 | Sajib Jahan | Masihur Maruf |

### **1. Introduction**

This document provides the technical specifications for the GoCloak project, an identity and access management (IAM) solution inspired by Keycloak, implemented in Go (Golang). The system aims to provide features like multi-tenancy, REST APIs for managing realms, clients, users, roles, and permissions, and an internal admin panel for managing these entities.

The document outlines the architecture, component design, and implementation strategies necessary for the development of GoCloak.

### **2. System Overview**

GoCloak will have the following core components:

**Key Goals**

* **High Performance**: The system should be capable of handling a high load, with the ability to support up to 100,000 concurrent sessions per realm.
* **Extensibility**: A modular architecture to support future integrations like external identity providers, MFA, and audit logs.
* **Ease of Deployment**: A simple deployment model using Docker and Kubernetes.

### **3. System Architecture**

**3.1. High-Level Architecture**

The GoCloak system architecture consists of the following key components:

* **Frontend (Admin Panel)**: A web-based UI for internal administrators to manage realms, users, and other IAM-related activities. This communicates with the backend via REST APIs.
* **Backend (REST API)**: Exposes endpoints for managing realms, users, clients, roles, permissions, and authentication flows. The backend service communicates with the database and handles authentication logic.
* **Database**: A **PostgreSQL** database (or **SQLite** in dev mode) will store the necessary data for realms, users, roles, permissions, and client applications.
* **Token Service**: Implements token management, including the generation, validation, and introspection of JWT tokens for OAuth2/OIDC compliance.
* **Security**: The backend will enforce HTTPS, and all sensitive data will be stored securely (e.g., bcrypt for password storage). Authentication will be token-based using JWT.

**3.2. System Components**

1. **API Gateway**: Handles incoming API requests, routing them to the appropriate service or endpoint (authentication, user management, etc.).
2. **Authentication Server**: Manages OAuth2/OIDC flows (token issuance, introspection, user info, etc.).
3. **User and Client Management**: Handles operations like creating, updating, deleting, and listing users and clients.
4. **Role & Permission Service**: Manages roles, permissions, and assignments for users and clients.
5. **Database Layer**: PostgreSQL (for production) or SQLite (for development), storing realm, user, client, and role data.
6. **Admin Panel**: A front-end built using **HTMX** and **Tailwind** to manage internal operations. This will interact with the backend through internal APIs.

### **4. Database Design**

The database schema for GoCloak consists of several tables related to realms, users, roles, and clients.

**4.1. Key Tables**

1. **Realms**
   * **id: Unique identifier**
   * **name: Realm name**
   * **description: Realm description**
   * **created\_at: Timestamp of realm creation**
   * **updated\_at: Timestamp of last update**
2. **Users**
   * **id: Unique identifier**
   * **realm\_id: Foreign key to the Realms table**
   * **username: User's username**
   * **password\_hash: Encrypted password (bcrypt or Argon2)**
   * **email: User's email address**
   * **roles: List of roles assigned to the user**
3. **Clients**
   * **id: Unique identifier**
   * **realm\_id: Foreign key to the Realms table**
   * **name: Client application name**
   * **client\_secret: Encrypted client secret**
   * **redirect\_uris: Allowed redirect URIs for OAuth2**
   * **grants: OAuth2 grants (e.g., authorization code, client credentials)**
4. **Roles**
   * **id: Unique identifier**
   * **realm\_id: Foreign key to the Realms table**
   * **name: Role name**
   * **permissions: List of permissions associated with the role**
5. **Tokens**
   * **id: Unique identifier**
   * **user\_id: Foreign key to the Users table**
   * **access\_token: JWT access token**
   * **refresh\_token: JWT refresh token**
   * **created\_at: Timestamp of token creation**
   * **expires\_at: Timestamp of token expiration**

### **5. API Design**

The following endpoints will be exposed by GoCloak for managing realms, clients, users, and authentication.

**5.1. Realm Management**

* GET /realms: List all realms (admin scope)
* POST /realms: Create a new realm
* GET /realms/{realm}: Get realm configuration
* PUT /realms/{realm}: Update realm configuration
* DELETE /realms/{realm}: Delete realm

**5.2. User Management**

* POST /realms/{realm}/users: Create a user
* GET /realms/{realm}/users/{id}: Get user details
* PUT /realms/{realm}/users/{id}: Update user
* DELETE /realms/{realm}/users/{id}: Delete user

**5.3. Client Management**

* POST /realms/{realm}/clients: Register a new client
* GET /realms/{realm}/clients: List all clients
* PUT /realms/{realm}/clients/{id}: Update client
* DELETE /realms/{realm}/clients/{id}: Delete client

**5.4. Token Management**

* POST /realms/{realm}/protocol/openid-connect/token: OAuth2 token endpoint
* POST /realms/{realm}/protocol/openid-connect/token/introspect: Token introspection
* GET /realms/{realm}/protocol/openid-connect/userinfo: Fetch user info from token

### **6. Security Design**

6.1. Authentication & Authorization

* The system will use JWT (JSON Web Tokens) for user authentication and authorization.
* OAuth2 and OpenID Connect will be supported for secure token-based login and authentication.
* Secure password storage (bcrypt or Argon2) will be used for user credentials.

6.2. Communication

* All external communications (API requests) will be encrypted using HTTPS.
* CSRF protection will be enforced for all internal API calls from the Admin Panel.

6.3. Permissions & Roles

* Role-based access control (RBAC) will be implemented for both users and clients.
* The admin panel will have granular permissions, allowing different users to have different access levels.

### **7. System Deployment**

GoCloak will be deployed in **Docker** containers for easy portability and scalability. The deployment process will include:

* **Docker Images** for each component (backend, database, and admin panel)
* Optional **Helm charts** for Kubernetes deployment
* **CI/CD pipelines** using **GitHub Actions** or **GitLab CI** for automated testing, building, and deployment.

### **8. Testing Strategy**

The testing strategy will include the following:

* **Unit Tests**: For individual components (e.g., API controllers, service functions).
* **Integration Tests**: To test the interaction between components (e.g., API with the database).
* **Load Testing**: To verify the system can handle a high number of concurrent sessions.
* **Security Testing**: To ensure the system is secure from vulnerabilities (e.g., penetration testing).
* **Acceptance Testing**: To verify that all user stories and requirements are met.

### **9. Performance & Scalability**

GoCloak is designed to scale horizontally with stateless services that can be replicated across multiple nodes. Key performance considerations include:

* **Database indexing** for fast queries, particularly on realm\_id, user\_id, and client\_id.
* **Caching** for frequently accessed data like roles and user details.
* **Load balancing** for distributing requests evenly across multiple instances of the backend service.

### **10. Conclusion**

This document provides the technical foundation for the development of GoCloak. By following the outlined architecture, design, and security considerations, GoCloak will deliver a high-performance, secure, and extensible IAM solution.