Documentation for JWT Authentication Service

Overview

The main objective of this task was to create and maintain a secure and efficient JWT authentication system across three microservices: login-service, kyc-service, and user-profile-service. The implementation focused on token-based authentication, ensuring seamless user interaction while maintaining data security, reliability, and scalability.

The task also included integrating centralized error monitoring with Sentry.io and leveraging PostgreSQL stored procedures for secure database operations. Common functionalities were modularized into shared files to maintain a clean and maintainable codebase.

**System Design and Implementation**

**1. JWT Authentication**

* **Purpose**: To establish a secure mechanism for authenticating users across all services using JSON Web Tokens (JWT).
* **JWT Workflow**:
  1. **Token Generation**:
     + A JWT token and session ID is generated upon successful OTP verification (via Twilio) in the login-service.
     + The JWT is signed using SigningMethodHS256 with a secret key and contains a payload that includes user-specific information and an expiration time.
  2. **Token Validation**:
     + Each subsequent request to kyc-service or user-profile-service requires the JWT for validation.
     + The token is validated using the same secret key to decrypt the payload and verify its integrity.
     + Expired or invalid tokens are rejected, ensuring secure access control.
* **Session Management**:
  1. Upon OTP verification, a session ID is generated and stored in the PostgreSQL database alongside the JWT.
  2. Sessions are validated during user interactions to ensure that the user remains authenticated throughout their activity.

**2. Login-Service**

* **Twilio OTP Verification**:
  + Used Twilio APIs to send and verify OTPs for user authentication.
  + Upon successful verification, a JWT and session ID are generated for the user and stored securely in the database.

**3. KYC-Service**

* **JWT Validation**:
  + Every KYC request is authenticated by validating the JWT token.
  + Data integrity and security are ensured by linking all operations to authenticated tokens.
* **Data Storage**:
  + After successful JWT validation, KYC data is securely stored in the database using stored procedures.

**4. User-Profile-Service**

* **User Data Retrieval**:
  + User profile data is fetched only after successful JWT validation.
  + This ensures authorized access to sensitive user data while maintaining the reliability and security of the process.

**Centralized Configuration and Code Reusability**

* **Shared Files**:  
  Common functionalities like database connections, Sentry initialization, and configuration management were modularized into a shared folder to avoid code duplication and promote reusability across all services.
  + **Database**: Centralized database connection logic was implemented to streamline PostgreSQL interaction.
  + **Sentry Integration**: A shared configuration for Sentry.io was utilized to provide consistent error tracking across services.
  + **Environment Variables**: Sensitive data and configurations were securely stored in .env files and accessed via a configuration module to prevent hardcoding and maintain consistent setups across environments.

**Error Monitoring with Sentry.io**

* Integrated Sentry.io into all three microservices (login-service, kyc-service, and user-profile-service).
* Real-time error tracking and monitoring were implemented to ensure quick debugging and resolution of issues.
* Centralized Sentry configuration provided a unified approach for error management across services.

**Database Management with PostgreSQL**

* Used PostgreSQL as the primary database for session and user data management.
* Created three tables for storing details securely
  + Users – contains the user’s phone\_number
  + Sessions – contains the session related data (for e.g. jwt\_token, sessionID and expiration time)
  + Kyc\_data – contains kyc related data in this tables.
  + Phone\_number and userID is foreign key in the sessions and kyc\_data table reference from users table.
* Designed stored procedures for key database operations:
  + **Session Management**: Created and managed sessions securely using stored procedures, eliminating the need for raw SQL queries within the code.
  + **Data Storage and Retrieval**: Securely stored and retrieved user and KYC data using parameterized stored procedures to avoid SQL injection and ensure efficient database interaction.
* GORM was used for seamless database interaction and query handling while maintaining flexibility for advanced operations.

**Challenges and Solutions**

1. **Token Management Across Services**:
   * Challenge: Ensuring secure and consistent handling of JWT tokens across all microservices.
   * Solution: Standardized token signing and validation using a common secret key and implementing clear error handling for expired or invalid tokens.
2. **Database Query Handling**:
   * Challenge: Avoiding raw SQL queries in the code to ensure security and maintainability.
   * Solution: Implemented stored procedures for all critical database operations and used GORM for streamlined database interaction.

**Conclusion**

The JWT authentication system was successfully implemented across three microservices, ensuring secure, scalable, and efficient user authentication and session management. By centralizing common functionalities, integrating Sentry for error monitoring, and leveraging stored procedures for database operations, the project achieved high reliability and maintainability.

This implementation sets a solid foundation for scalable and secure user authentication, paving the way for future enhancements and integrations.