CMIPS Security Enhanced Architecture Proposal

Document Overview

This document provides a detailed analysis of the enhanced CMIPS security architecture.

System Overview

Our proposed solution is an advanced workflow management platform designed to streamline **timesheet processing, approvals, and user workflows** within complex organizational structures. The system supports **granular field-level authorization** and **role-based access control (RBAC)** to ensure that each user can access only the information and actions relevant to their role.

Security and Compliance

Built with a strong focus on **data security** and **regulatory compliance**, the platform leverages **Keycloak Identity and Access Management (IAM)** for centralized authentication and authorization. This integration allows for **dynamic, configurable access control**, enabling administrators to modify permissions and workflows directly through Keycloak — without requiring backend code changes.

Key Benefits

- **Secure and Scalable:** Robust protection of sensitive data aligned with compliance standards.

- **Configurable Authorization:** Role and field-level access rules can be adjusted dynamically.

- **Streamlined Workflows:** Simplified management of timesheets, approvals, and user actions.

- **Centralized Access Control:** Unified user and permission management through Keycloak IAM.

🎯 Core Features Implemented

1. Multi-Level Field Authorization

- Granular **field-level permissions** for operations such as *read, create, edit, delete, approve,* and *reject*.

- **Dynamic field filtering** based on user roles, scopes, and contextual parameters.

- **Configurable field visibility** and access control managed directly through **Keycloak attributes**, allowing administrators to adjust permissions without code changes.

2. Role-Based Access Control (RBAC)

RBAC is a **security model** that assigns permissions based on user roles rather than individuals. Each role defines **what actions a user can perform** and **what data they can access**, ensuring **consistent, secure, and easy-to-manage access control** across the system.

Example in CMIPS:

- **Provider:** Manages their own timesheets.

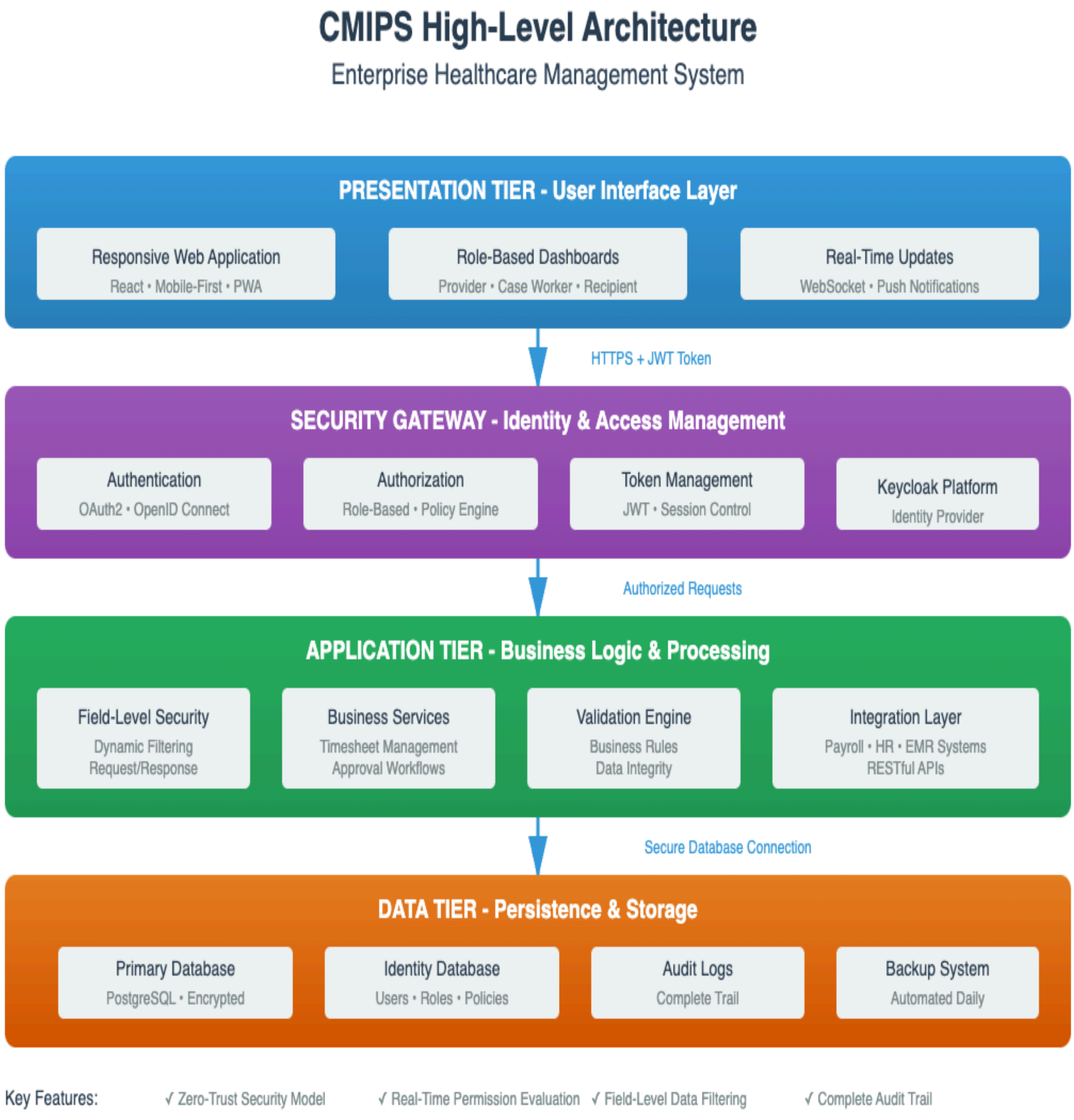
- **Case Worker:** Reviews and approves/rejects timesheets.

- **Recipient:** Views only their own information.

CMIPS High-Level Architecture Overview

Architecture Pattern: Multi-Tier Architecture

CMIPS adopts a 4-tier enterprise architecture pattern designed to separate concerns, ensure scalability, and maintain robust security across all layers. This architecture aligns with industry best practices for healthcare systems handling sensitive data.



Tier 1: Presentation Layer (Client Tier)

Purpose

The presentation layer serves as the user-facing component, responsible for displaying information and capturing user input.

Components

- **Responsive Web Application**: Built using React with a mobile-first design and Progressive Web App (PWA) capabilities.
- **Role-Based Dashboards**:
    - *Provider*: Timesheet creation, submission, and history tracking.
    - *Case Worker*: Approval queue, review tools, and reporting.
    - *Recipient*: Service visibility and history tracking.
    - *Admin*: User management, configuration, and analytics.
- **Dynamic UI Rendering**: UI elements adapt based on user permissions; buttons and form fields appear/hide automatically without hardcoded frontend checks.
- **Real-Time Features**: WebSocket connections for instant updates, push notifications, live status updates, and instant feedback.

Key Characteristics

- Stateless architecture with token-based communications using JWT.

Tier 2: Security Gateway Layer (Identity & Access Management)

Purpose

Acts as a security checkpoint, ensuring only authenticated and authorized users can access the system.

Components

- **Authentication Service (Keycloak)**:
    - OAuth2 / OpenID Connect, username/password authentication.

    - Multi-factor authentication (MFA), session management, and Single Sign-On (SSO).
- **Authorization Engine**:
    - Role-Based Access Control (RBAC).
    - Resource-based permissions and scope validation.
    - Policy evaluation engine for dynamic permission calculation.
- **Token Management**:
    - JWT token issuance, validation, expiration, refresh, and revocation.
- **Keycloak Platform**:
    - Centralized identity provider with role management, password policies, and audit logging.

Security Features

- Zero Trust model (every request validated).
- Token encryption, session control

Tier 3: Application Layer (Business Logic Tier)

Purpose

Handles all business logic, data processing, and orchestration, acting as the brain of the system.

Components

- **API Gateway & Security Filter**: Request routing, rate limiting, logging, error handling, and CORS management.
- **Authorization Middleware**: Permission evaluation, resource access control, scope validation, policy enforcement, and caching.
- **Field-Level Security Engine**: Dynamic request/response field filtering based on role and operation, fully configuration-driven.
- **Business Logic Services**:
    - *Timesheet Management*: CRUD operations, status workflow, validation, and ownership checks.
    - *Approval Workflow Service*: Approval/rejection, supervisor comments, notifications, and status transitions.
    - *Validation Engine*: Business rule enforcement, data integrity checks, and duplicate detection.
- **Integration Layer**:
    - Payroll system export, HR system sync, EMR integration, and RESTful APIs.

Key Characteristics

- Stateless and microservices-ready.
- Event-driven architecture for notifications and integrations.
- Highly available with multiple instances behind a load balancer.

Performance Optimizations

- Connection pooling (DB), permission caching (Redis), query optimization, and asynchronous processing.

Tier 4: Data Layer (Persistence Tier)

Purpose

Stores, retrieves, and manages all persistent data, ensuring durability, consistency, and recoverability.

Components

- **Primary Database (PostgreSQL)**: Stores timesheets, case management, payments, and user profiles; encrypted (AES-256), ACID compliant, optimized for queries and backup.
- **Identity Database (Keycloak Database)**: Stores user credentials, roles, permissions, and policies; isolated and encrypted.
- **Audit Log Storage**: Tracks all access and changes, immutable, tamper-proof, retained for 7 years.
- **Backup System**: Automated daily and incremental backups, encrypted, and tested for recovery.

Cross-Cutting Concerns

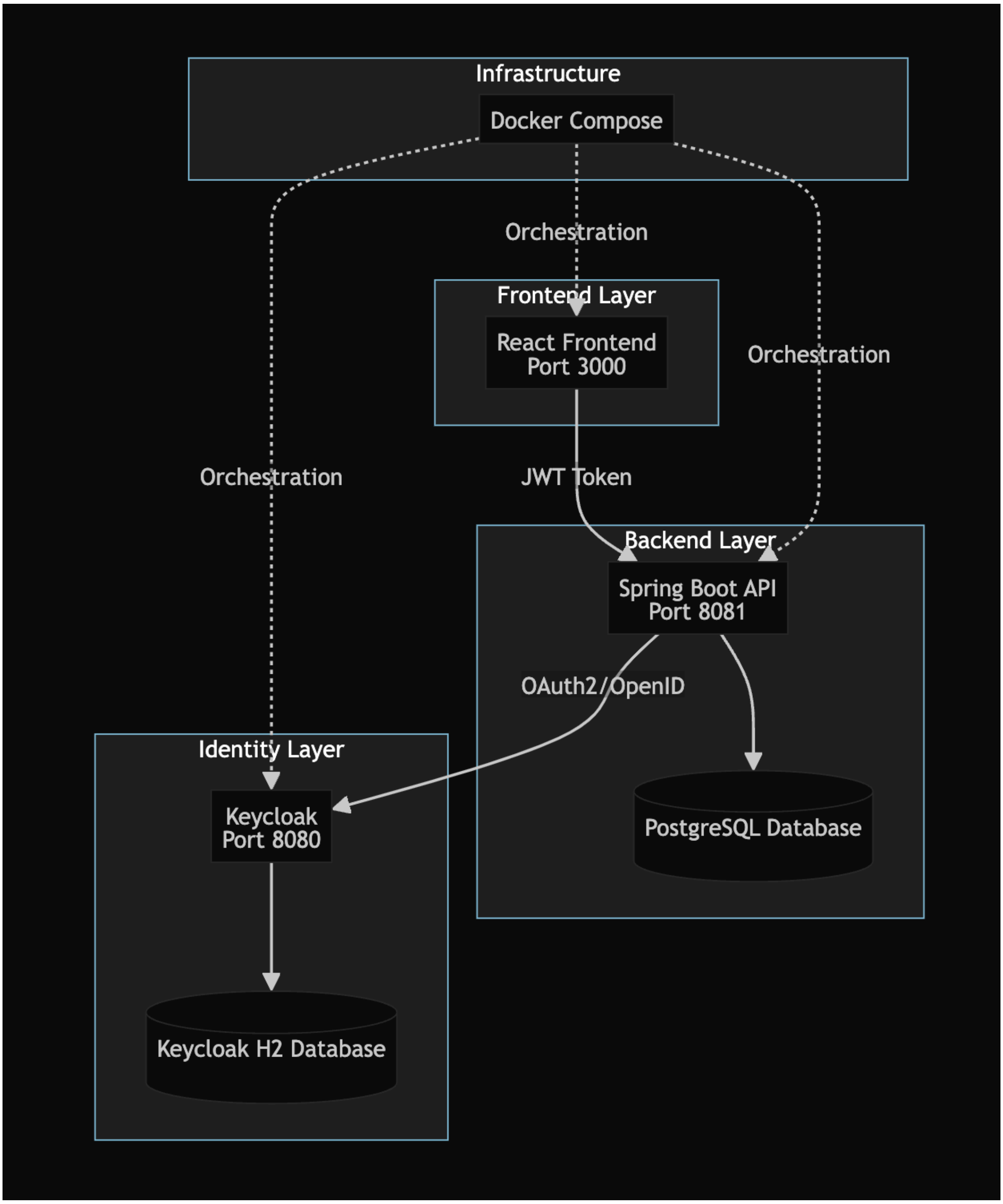
Communication Between Tiers

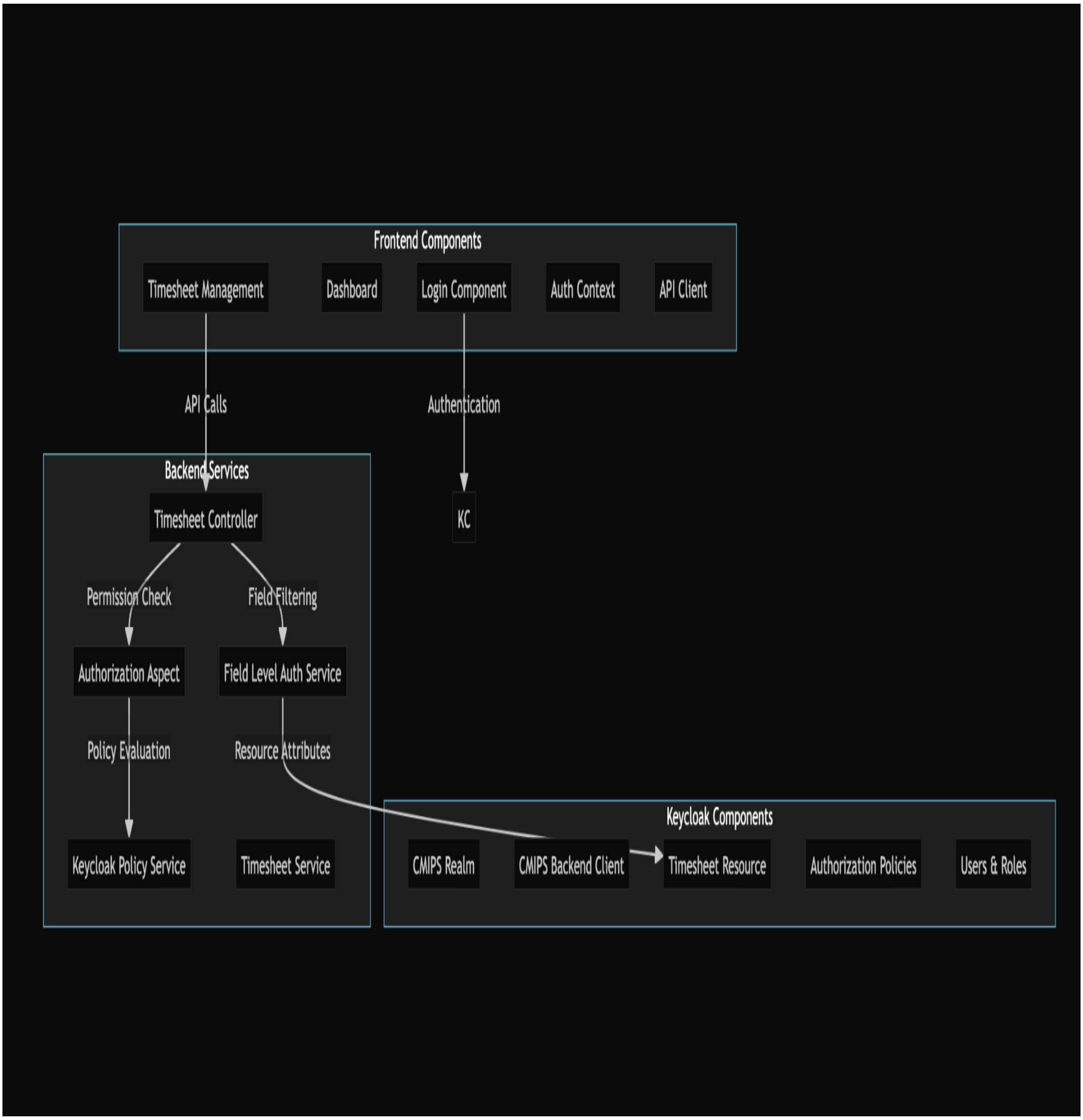
- **Presentation ↔ Security Gateway**: HTTPS/TLS, JWT Bearer Tokens, JSON.
- **Security Gateway ↔ Application**: Internal network, token validation.
- **Application ↔ Data**: Encrypted database connections, connection pooling.

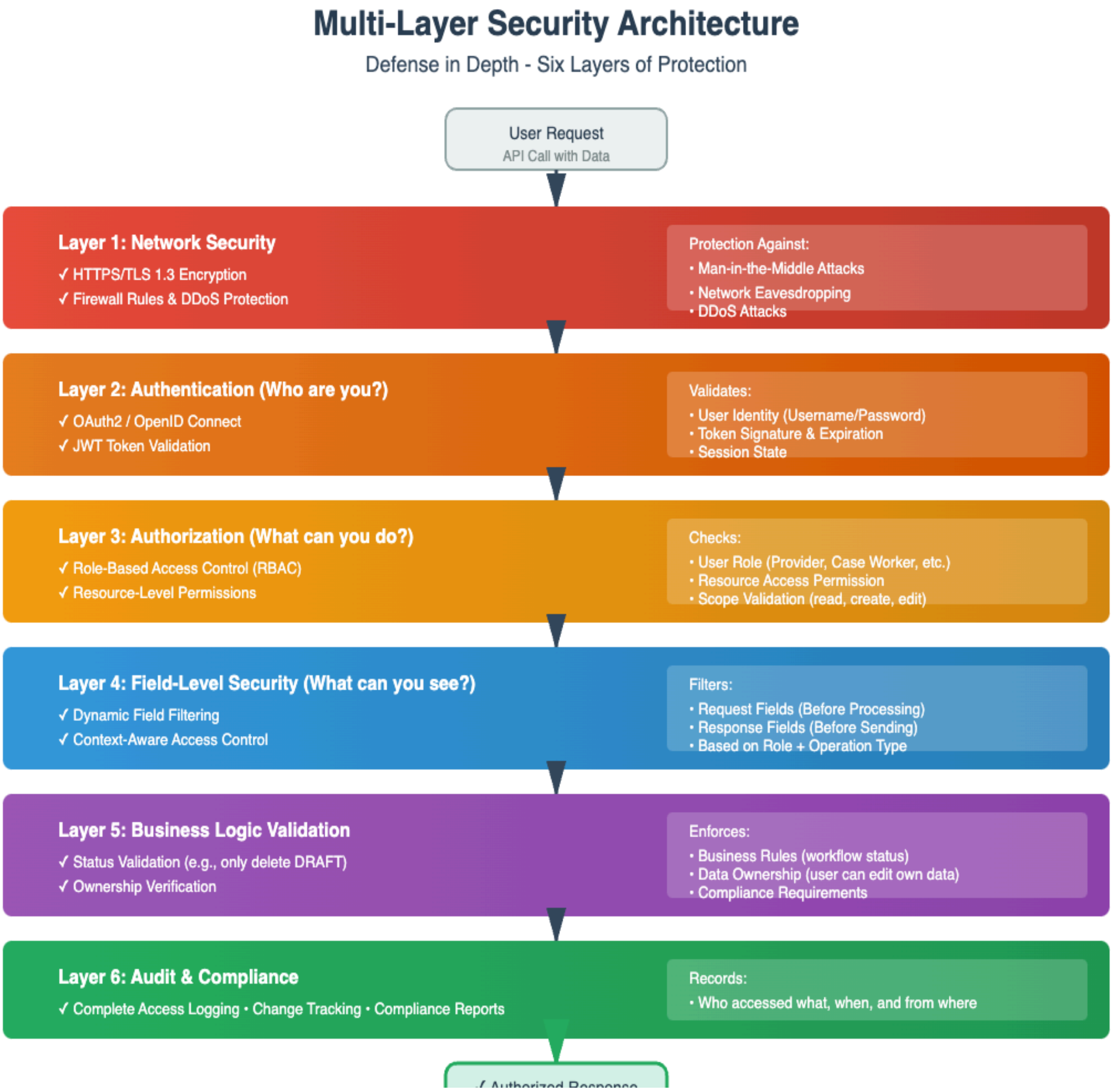
Monitoring & Observability

- **Application**: Response time, error rates, CPU/memory usage.
- **Security**: Failed logins, unauthorized access, token expiration.
- **Business**: Timesheet submissions, approval turnaround, usage metrics.

Detailed Component Architecture




Frontend Components
Timesheet Management
Dashboard
Login Component
Auth Context
API Client
API Calls
Authentication
Backend Services
Timesheet Controller
KC
Permission Check
Field Filtering
Authorization Aspect
Field Level Auth Service
Policy Evaluation
Resource Attributes
Keycloak Components
Keycloak Policy Service
Timesheet Service
CMIPS Realm
CMIPS Backend Client
Timesheet Resource
Authorization Policies
Users & Roles



Multi-Layer Security Architecture

Defense in Depth - Six Layers of Protection

User Request
API Call with Data

Layer 1: Network Security
✓ HTTPS/TLS 1.3 Encryption
✓ Firewall Rules & DDoS Protection

Protection Against:
- Man-in-the-Middle Attacks
- Network Eavesdropping
- DDoS Attacks

Layer 2: Authentication (Who are you?)
✓ OAuth2 / OpenID Connect
✓ JWT Token Validation

Validates:
- User Identity (Username/Password)
- Token Signature & Expiration
- Session State

Layer 3: Authorization (What can you do?)
✓ Role-Based Access Control (RBAC)
✓ Resource-Level Permissions

Checks:
- User Role (Provider, Case Worker, etc.)
- Resource Access Permission
- Scope Validation (read, create, edit)

Layer 4: Field-Level Security (What can you see?)
✓ Dynamic Field Filtering
✓ Context-Aware Access Control

Filters:
- Request Fields (Before Processing)
- Response Fields (Before Sending)
- Based on Role + Operation Type

Layer 5: Business Logic Validation
✓ Status Validation (e.g., only delete DRAFT)
✓ Ownership Verification

Enforces:
- Business Rules (workflow status)
- Data Ownership (user can edit own data)
- Compliance Requirements

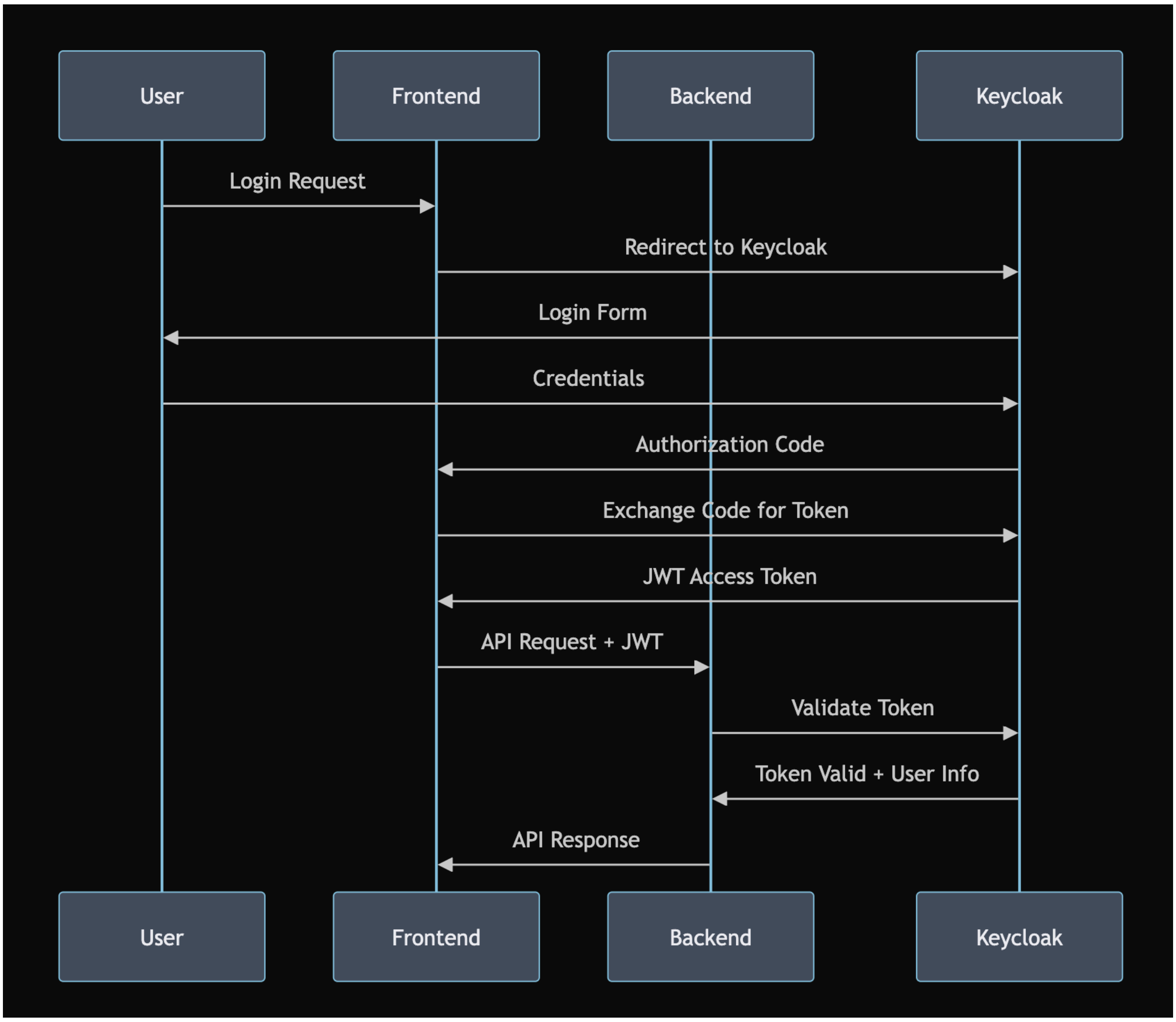
Layer 6: Audit & Compliance
✓ Complete Access Logging • Change Tracking • Compliance Reports

Records:
- Who accessed what, when, and from where

✓ Authorized Response

CMIPS Authentication and Authorization

Complete Authentication Flow:

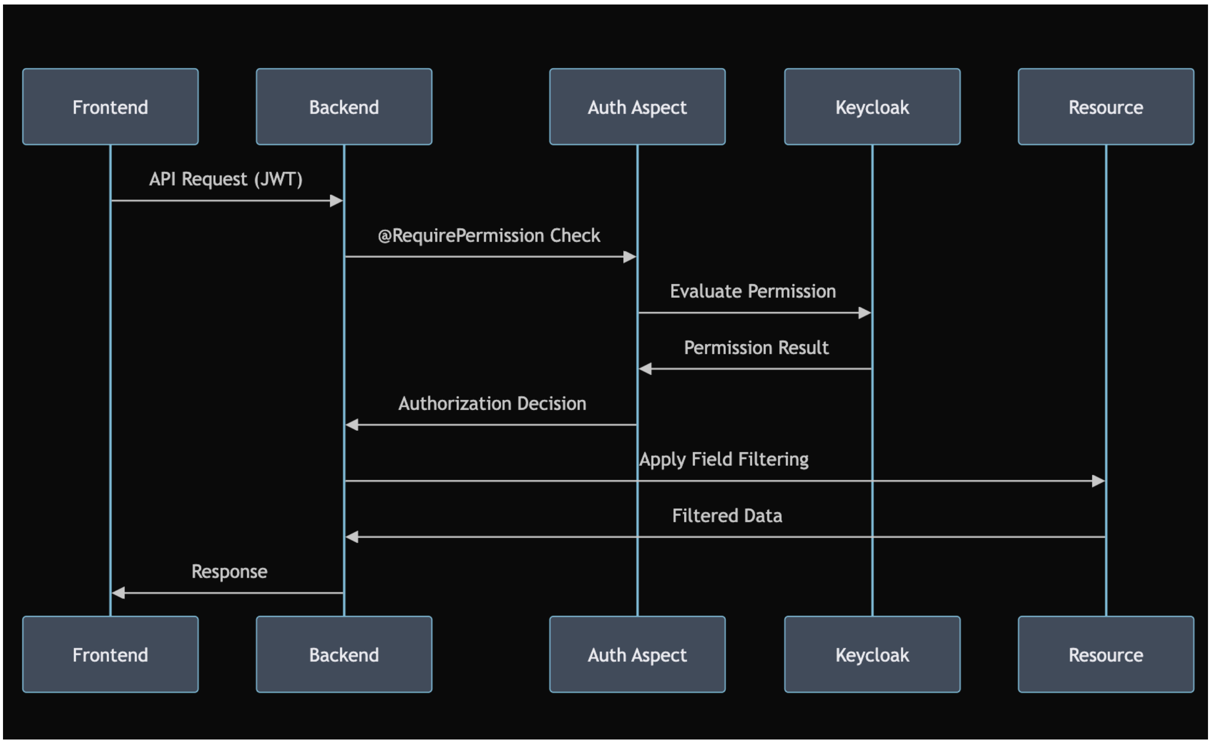


1. User enters credentials in the frontend.
2. Frontend sends login request to Keycloak.
3. Keycloak validates credentials against the database.
4. Keycloak generates JWT token with user info and roles.
5. Frontend stores the token and sends it with API requests.
6. Spring Boot validates JWT signature using Keycloak's public key.
7. Spring Boot extracts user ID and roles from the token.
8. Spring Boot calls Keycloak Authorization Services.
9. Keycloak evaluates policies and permissions.
10. Keycloak returns authorization decision.
11. Spring Boot enforces Keycloak's decision.
12. Controller executes business logic if authorized.
13. Response sent back to the frontend.

Security Features

- Password hashing (bcrypt) ✅
- JWT token signatures (RSA-SHA256) ✅
- Token expiration (5 minutes) ✅
- Role-based access control ✅
- CORS protection ✅
- HTTPS-ready configuration ✅
- Session management ✅

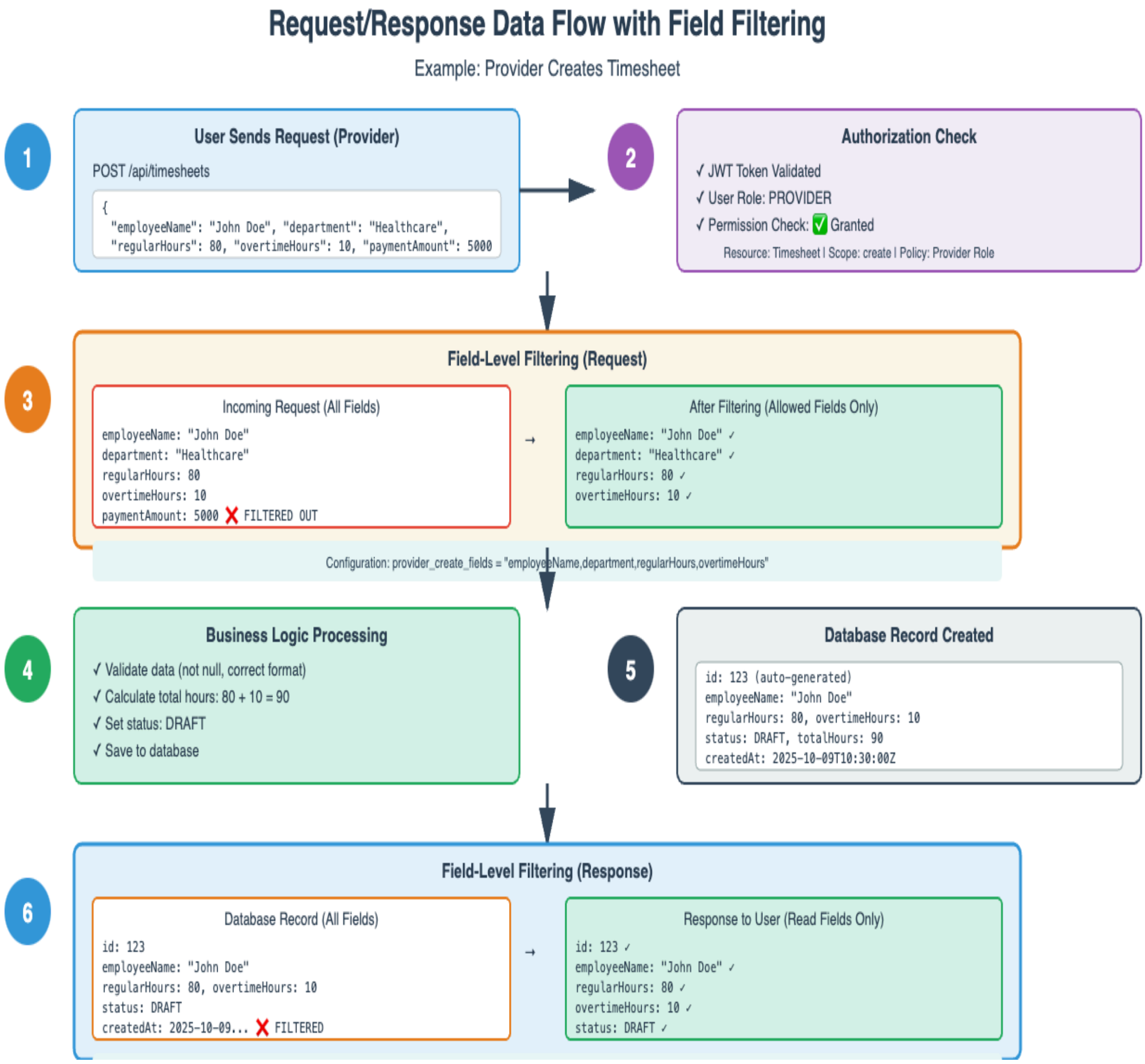
Authorization Flow:



1. User makes API request with JWT token.
2. Spring Boot extracts access token from the request.
3. Spring Boot calls Keycloak Authorization Services.
4. Keycloak evaluates policies and permissions.
5. Keycloak returns authorization decision.
6. Spring Boot enforces Keycloak's decision.

Detailed Authorization Workflow

User Request → JWT Token → Spring Security → Keycloak Public Key Validation
↓
@RequirePermission → AuthorizationAspect → KeycloakPolicyEvaluationService
↓
Keycloak Authorization Services → Policy Evaluation → Authorization Decision
↓
Backend Business Logic (if authorized) → Response



Request/Response Data Flow with Field Filtering

Example: Provider Creates Timesheet

User Sends Request (Provider)

POST /api/timesheets

```
{
  "employeeName": "John Doe", "department": "Healthcare",
  "regularHours": 80, "overtimeHours": 10, "paymentAmount": 5000
```

Authorization Check

✓ JWT Token Validated
✓ User Role: PROVIDER
✓ Permission Check: ✅ Granted
Resource: Timesheet | Scope: create | Policy: Provider Role

Field-Level Filtering (Request)

Incoming Request (All Fields)

```
employeeName: "John Doe"
department: "Healthcare"
regularHours: 80
overtimeHours: 10
paymentAmount: 5000 ❌ FILTERED OUT
```

After Filtering (Allowed Fields Only)

```
employeeName: "John Doe" ✓
department: "Healthcare" ✓
regularHours: 80 ✓
overtimeHours: 10 ✓
```

Configuration: provider_create_fields = "employeeName,department,regularHours,overtimeHours"

Business Logic Processing

✓ Validate data (not null, correct format)
✓ Calculate total hours: 80 + 10 = 90
✓ Set status: DRAFT
✓ Save to database

Database Record Created

```
id: 123 (auto-generated)
employeeName: "John Doe"
regularHours: 80, overtimeHours: 10
status: DRAFT, totalHours: 90
createdAt: 2025-10-09T10:30:00Z
```

Field-Level Filtering (Response)

Database Record (All Fields)

```
id: 123
employeeName: "John Doe"
regularHours: 80, overtimeHours: 10
status: DRAFT
createdAt: 2025-10-09... ❌ FILTERED
```

Response to User (Read Fields Only)

```
id: 123 ✓
employeeName: "John Doe" ✓
regularHours: 80 ✓
overtimeHours: 10 ✓
status: DRAFT ✓
```

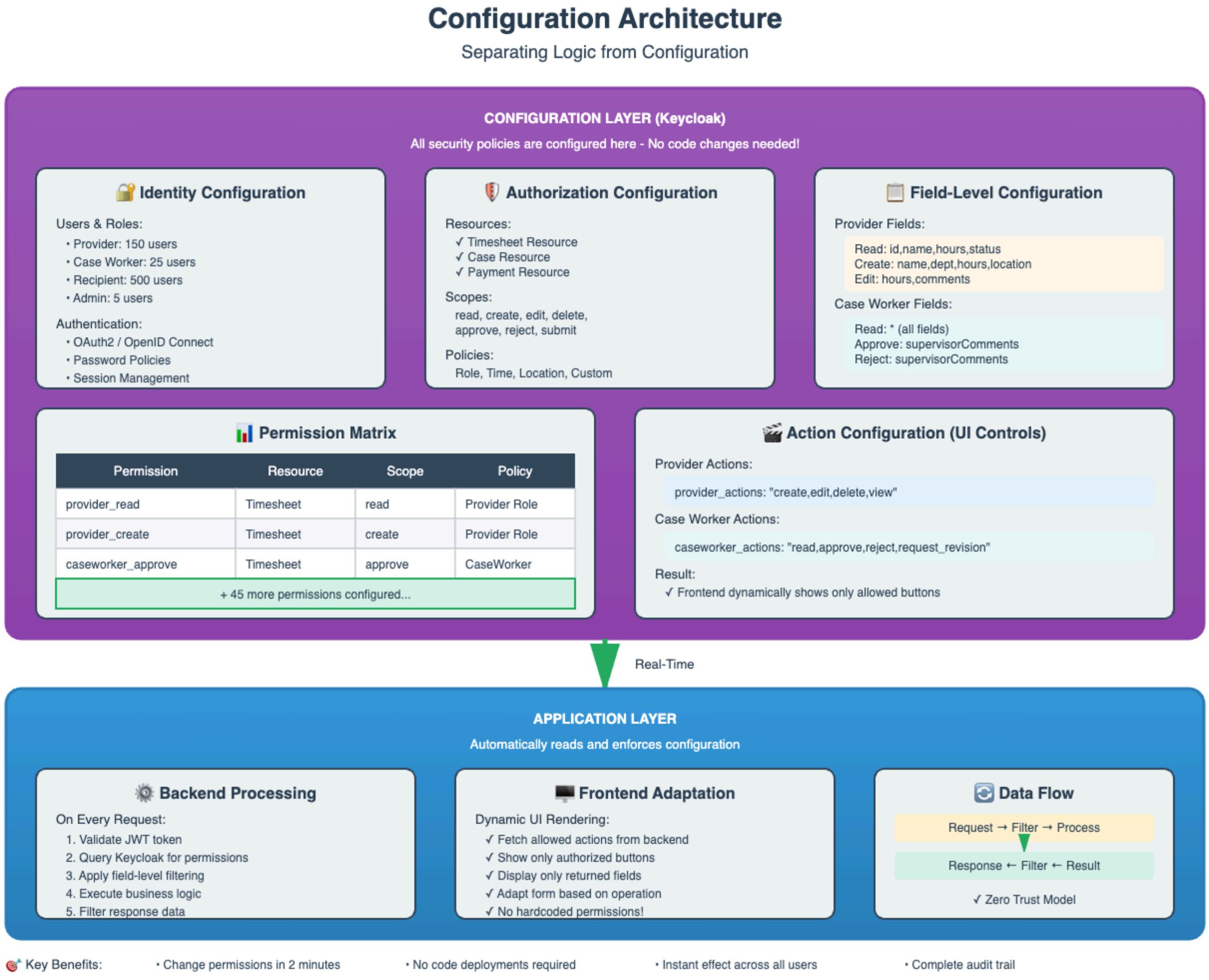
Policy Evaluation Engine

When a user requests access, Keycloak's policy evaluation engine:

1. Identifies the user's roles (from JWT token).
2. Finds matching policies (e.g., Case Worker Policy, Recipient Policy).
3. Evaluates permissions (which scopes are granted).
4. Returns authorization token with granted scopes.

Configuration Change Workflow

1. Admin creates ServiceNow ticket for config change.
2. Ticket includes current and proposed configuration.
3. Senior admin reviews and approves.
4. Approved ticket triggers Keycloak update.
5. Change logged and audited.



Configuration Architecture

Separating Logic from Configuration

CONFIGURATION LAYER (Keycloak)

All security policies are configured here - No code changes needed!

Identity Configuration

Users & Roles:
- Provider: 150 users
- Case Worker: 25 users
- Recipient: 500 users
- Admin: 5 users

Authentication:
- OAuth2 / OpenID Connect
- Password Policies
- Session Management

Authorization Configuration

Resources:
✓ Timesheet Resource
✓ Case Resource
✓ Payment Resource

Scopes:
read, create, edit, delete,
approve, reject, submit

Policies:
Role, Time, Location, Custom

Field-Level Configuration

Provider Fields:
Read: id,name,hours,status
Create: name,dept,hours,location
Edit: hours,comments

Case Worker Fields:
Read: * (all fields)
Approve: supervisorComments
Reject: supervisorComments

Permission Matrix

Permission	Resource	Scope	Policy
provider_read	Timesheet	read	Provider Role
provider_create	Timesheet	create	Provider Role
caseworker_approve	Timesheet	approve	CaseWorker
+ 45 more permissions configured...			

Action Configuration (UI Controls)

Provider Actions:
provider_actions: "create,edit,delete,view"

Case Worker Actions:
caseworker_actions: "read,approve,reject,request_revision"

Result:
✓ Frontend dynamically shows only allowed buttons

APPLICATION LAYER

Automatically reads and enforces configuration

Backend Processing

On Every Request:
1. Validate JWT token
2. Query Keycloak for permissions
3. Apply field-level filtering
4. Execute business logic
5. Filter response data

Frontend Adaptation

Dynamic UI Rendering:
✓ Fetch allowed actions from backend
✓ Show only authorized buttons
✓ Display only returned fields
✓ Adapt form based on operation
✓ No hardcoded permissions!

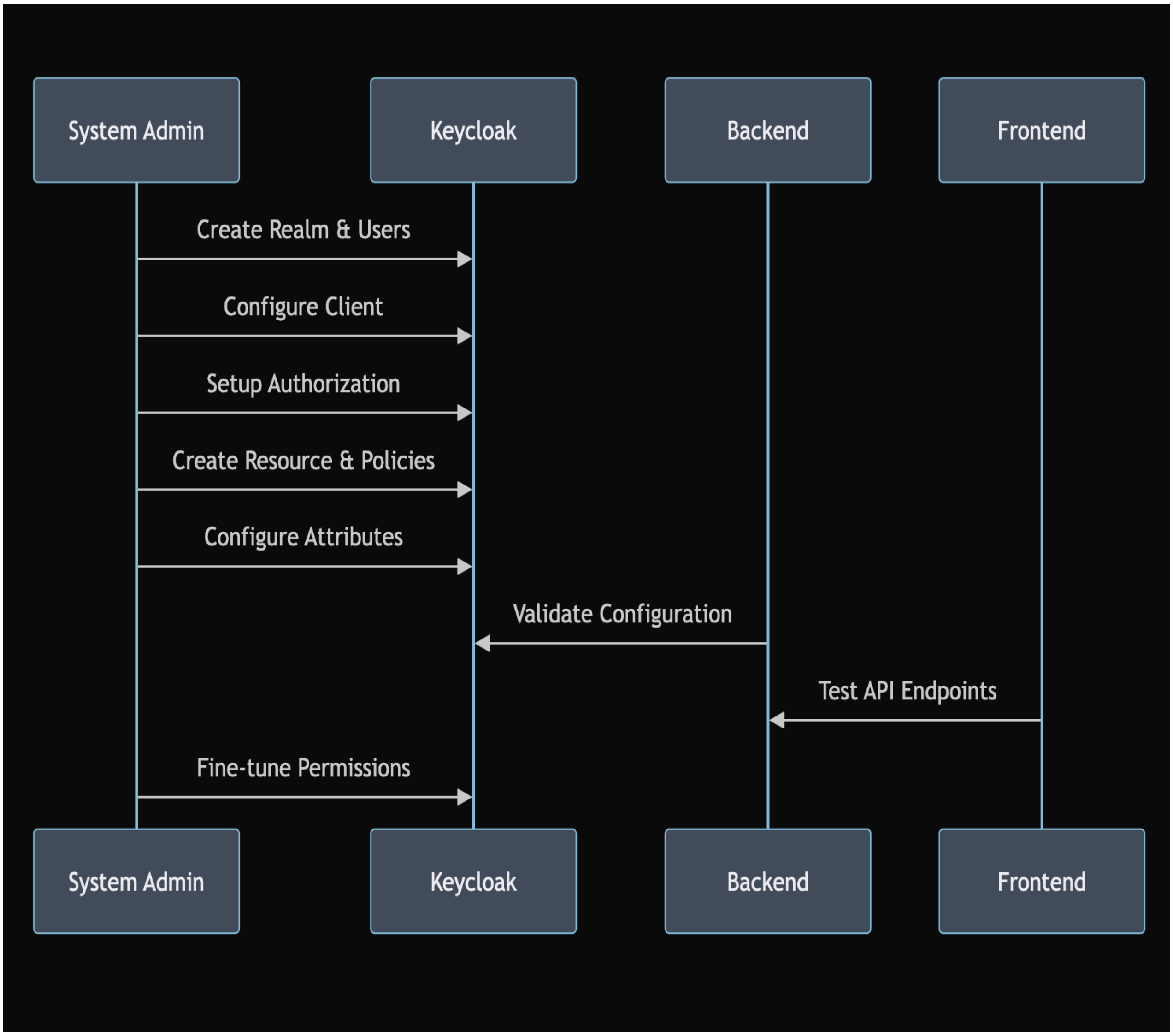
Data Flow

✓ Zero Trust Model

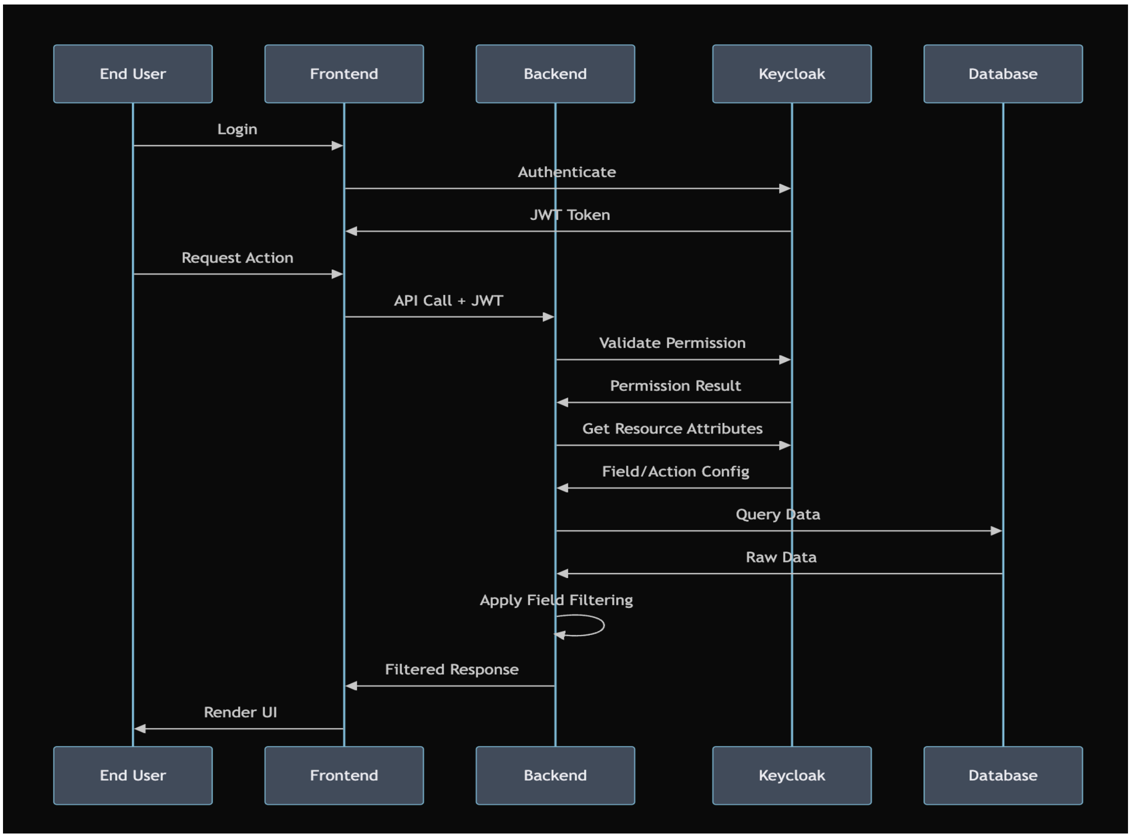
Key Benefits: • Change permissions in 2 minutes • No code deployments required • Instant effect across all users • Complete audit trail

Configuration Work Flow Diagrams:

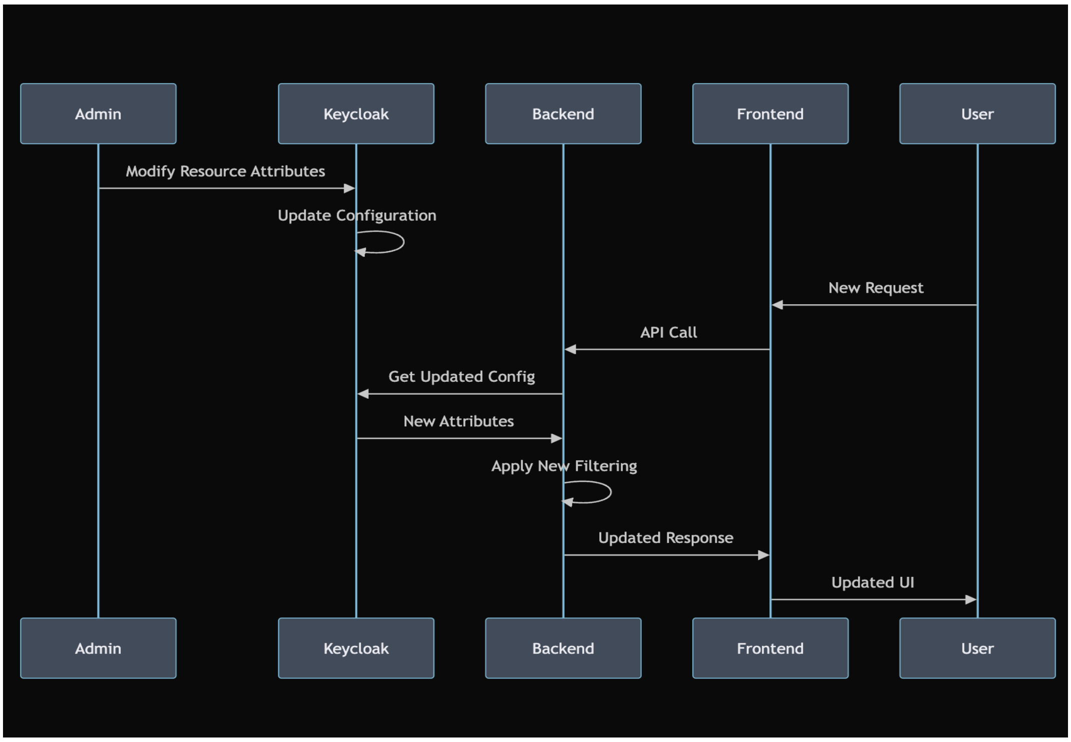
1.Initial Setup Flow:



2.Runtime Permission Flow:



3. Configuration Change Flow:



ServiceNow Ticketing & Keycloak Configuration Integration

Overview: ServiceNow + Keycloak Integration

Why Integrate ServiceNow with Keycloak?

- ServiceNow manages the approval workflow and change management
- Keycloak remains the configuration store and identity provider
- Integration ensures governed changes with proper approvals

Integration Architecture

Components Involved

- **ServiceNow**: IT Service Management platform
- **Keycloak**: Identity and Access Management
- **Integration Middleware**: REST API connector
- **Approval Workflow**: Multi-level approval process
- **Audit System**: Change tracking and compliance

Key Benefits

- ✅ **Governed Changes**: All configuration changes go through approval process
- ✅ **Audit Compliance**: Complete paper trail for compliance audits
- ✅ **Risk Management**: Changes reviewed before implementation
- ✅ **Rollback Capability**: Easy to revert approved changes
- ✅ **Automation**: Approved changes automatically applied
- ✅ **Documentation**: Self-documenting change management

Detailed Workflow Steps

1. Change Request Initiation

- **Admin Action**: Identifies need for Keycloak configuration change (e.g., new role, policy update, realm modification)
- **Ticket Creation**: Admin creates a ServiceNow Change Request ticket

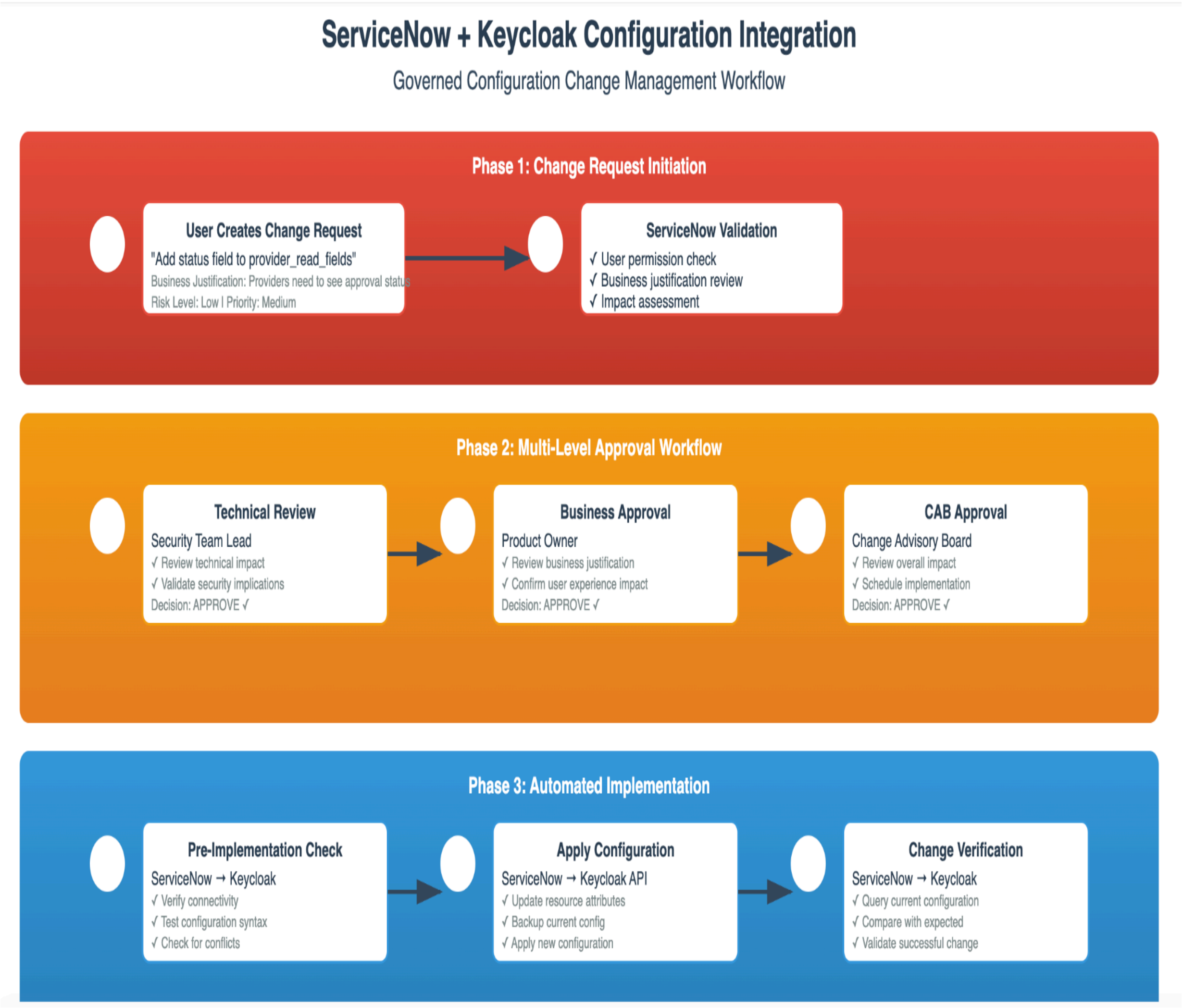
- **Ticket Contents**:
    - Current configuration state
    - Proposed configuration changes
    - Business justification
    - Risk assessment
    - Implementation plan
    - Rollback procedure

2. Multi-Level Approval Process

- **Level 1 Approval**: Direct manager or team lead reviews for technical validity
- **Level 2 Approval**: Senior security administrator reviews for compliance and risk
- **Approval Criteria**:
    - Technical feasibility
    - Security impact assessment
    - Compliance alignment
    - Business need validation

3. Validation & Audit

- **Pre-Implementation Validation**: Keycloak validates configuration syntax and references
- **Post-Implementation Verification**: Automated tests confirm successful application
- **Audit Trail**:
    - Who requested the change
    - Who approved the change
    - When the change was implemented
    - What exactly was changed
    - Configuration before/after comparison



ServiceNow + Keycloak Configuration Integration

Governed Configuration Change Management Workflow

Phase 1: Change Request Initiation

User Creates Change Request
"Add status field to provider_read_fields"
Business Justification: Providers need to see approval status
Risk Level: Low | Priority: Medium

ServiceNow Validation
✓ User permission check
✓ Business justification review
✓ Impact assessment

Phase 2: Multi-Level Approval Workflow

Technical Review
Security Team Lead
✓ Review technical impact
✓ Validate security implications
Decision: APPROVE ✓

Business Approval
Product Owner
✓ Review business justification
✓ Confirm user experience impact
Decision: APPROVE ✓

CAB Approval
Change Advisory Board
✓ Review overall impact
✓ Schedule implementation
Decision: APPROVE ✓

Phase 3: Automated Implementation

Pre-Implementation Check
ServiceNow → Keycloak
✓ Verify connectivity
✓ Test configuration syntax
✓ Check for conflicts

Apply Configuration
ServiceNow → Keycloak API
✓ Update resource attributes
✓ Backup current config
✓ Apply new configuration

Change Verification
ServiceNow → Keycloak
✓ Query current configuration
✓ Compare with expected
✓ Validate successful change

This integration ensures that all Keycloak configuration changes are properly governed, auditable, and compliant with enterprise change management standards while maintaining the flexibility and power of Keycloak's identity management capabilities.

Keycloak Authentication Load Test Report

1. Overview

This report summarizes the results of the Keycloak Authentication Endpoint Load Testing conducted using Apache JMeter.
The purpose of this test was to evaluate the scalability, stability, and response performance of Keycloak under increasing levels of concurrent authentication requests.

2. Test Objective

To validate the performance and reliability of Keycloak's authentication service across different user loads — from small-scale to large-scale concurrent sessions — and to identify any system limitations impacting scalability.

3. Test Environment

Component	Details
Testing Tool	Apache JMeter
Target System	Keycloak Authentication Endpoint
System Under Test (SUT)	macOS
Duration (for final test)	4 minutes 32 seconds (272 seconds)
Memory per JMeter Thread	~2 MB
JMeter Configuration	Each thread = 1 simulated user

4. Test Scenarios

A progressive testing approach was used to gradually increase the load and observe Keycloak's performance.

Test Round	Configured Users (Threads)	Execution Result	Success Rate	Observation
Test 1	100 users	All requests successful	✅ 100%	Stable, fast responses
Test 2	500 users	All requests successful	✅ 100%	No latency or errors
Test 3	1,000 users	All requests successful	✅ 100%	Excellent stability
Test 4	10,000 users	Executed 4,055 requests before system limit reached	✅ 100% (for executed requests)	macOS thread limitation caused test to stop

5. Summary of 10,000 User Test

Metric	Result	Interpretation
Total Requests Executed	4,055	Test stopped due to OS thread limit
Success Rate	100%	No authentication errors
Average Response Time	18 ms	Outstanding performance
Minimum Response Time	15 ms	Very low latency
Maximum Response Time	60 ms	Within expected range
Standard Deviation	1.51 ms	Highly consistent
Peak Throughput	20 requests/second	Excellent sustained performance
Actual Throughput	14.9 requests/second	Based on 272-second run time

6. System Limitation Analysis

The load test reached **4,056 concurrent users** before encountering a **thread creation failure** on macOS.
This issue was **not related to Keycloak**, but rather to **system-level resource constraints**.

Root Cause:

- macOS enforces limits on:
  - Maximum threads per process
  - Memory allocation per thread (≈2MB each)
- At 4,056 threads, approximately **8GB+ of memory** was consumed, preventing new thread creation.

Conclusion: Keycloak remained stable and responsive even as the test system reached its thread limit.

7. Response Time & Throughput Analysis

Response Time:

- **Average:** 18ms → Very fast
- **Range:** 15ms – 60ms
- **Consistency:** Standard deviation of 1.51ms confirms stable behavior

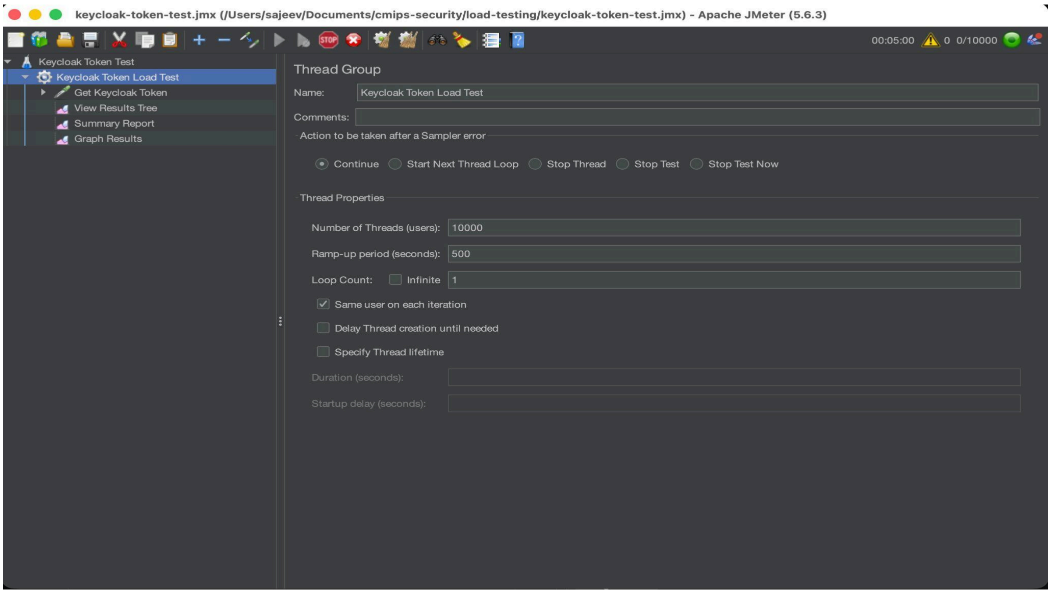
Throughput:

- **Peak:** 20 requests/second
- **Sustained Average:** 14.9 requests/second
- **Data Transfer:**
  - Received: 47.78 KB/sec
  - Sent: 6.47 KB/sec
  - Average Response Size: 2,446 bytes

8. Conclusion

The load testing results confirm that **Keycloak delivers exceptional performance and reliability** as an identity and access management solution.
Even under thousands of concurrent authentication requests, Keycloak maintained **stable, fast, and error-free** performance.
The only observed limitation originated from the **test environment**, not the Keycloak system.

Final Assessment:
✅ Keycloak is **highly efficient**, **scalable**, and **production-ready** for enterprise authentication workloads.



keycloak-token-test.jmx (/Users/sajeev/Documents/cmips-security/load-testing/keycloak-token-test.jmx) - Apache JMeter (5.6.3)

00:05:00 0 0/10000

- Keycloak Token Test
  - Keycloak Token Load Test
    - Get Keycloak Token
    - View Results Tree
    - Summary Report
    - Graph Results

Thread Group

Name: Keycloak Token Load Test

Comments:

Action to be taken after a Sampler error

● Continue ○ Start Next Thread Loop ○ Stop Thread ○ Stop Test ○ Stop Test Now

Thread Properties

Number of Threads (users): 10000

Ramp-up period (seconds): 500

Loop Count: □ Infinite 1

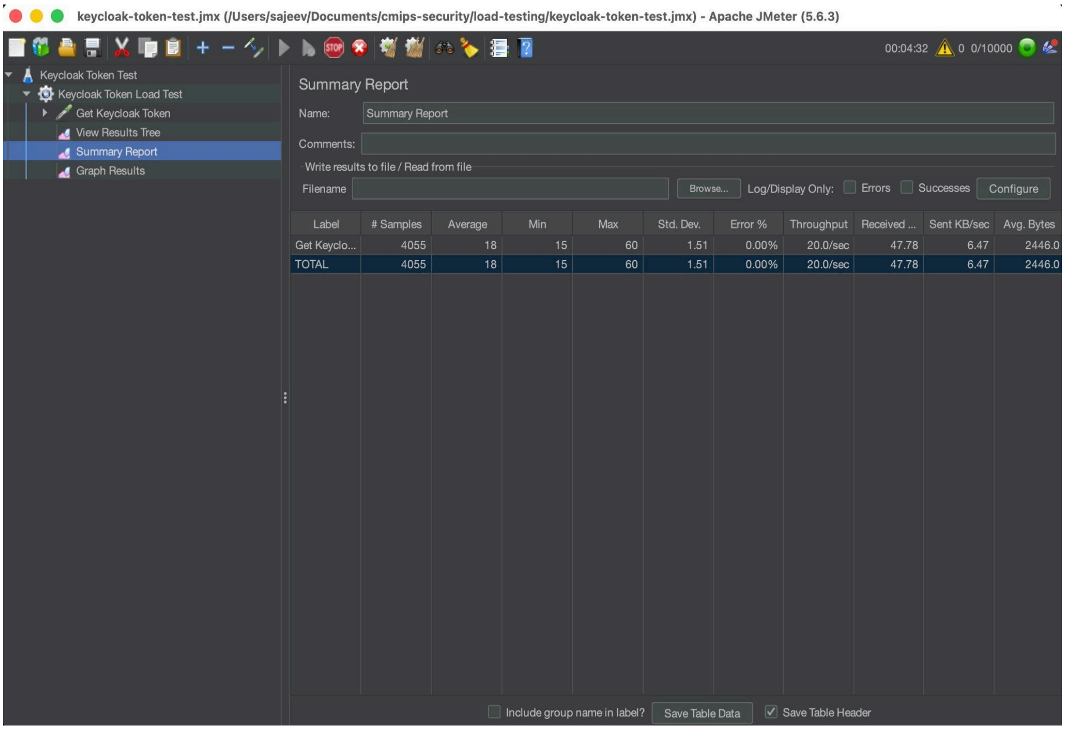
☑ Same user on each iteration

□ Delay Thread creation until needed

□ Specify Thread lifetime

Duration (seconds):

Startup delay (seconds):



keycloak-token-test.jmx (/Users/sajeev/Documents/cmips-security/load-testing/keycloak-token-test.jmx) - Apache JMeter (5.6.3)

00:04:32 0 0/10000

- Keycloak Token Test
  - Keycloak Token Load Test
    - Get Keycloak Token
    - View Results Tree
    - Summary Report
    - Graph Results

Summary Report

Name: Summary Report

Comments:

Write results to file / Read from file

Filename Browse... Log/Display Only: □ Errors □ Successes Configure

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received ...	Sent KB/sec	Avg. Bytes
Get Keyclo...	4055	18	15	60	1.51	0.00%	20.0/sec	47.78	6.47	2446.0
TOTAL	4055	18	15	60	1.51	0.00%	20.0/sec	47.78	6.47	2446.0

□ Include group name in label? Save Table Data ☑ Save Table Header