

# SOFTWARE REQUIREMENTS SPECIFICATION

**BMW IT HUB**

COS 301 | University of Pretoria

---

## **FIRE-FIGHTER ACCESS MANAGEMENT PLATFORM**

---

# Contents

|  |    |
|--|----|
| 1. Introduction                        | 3  |
| 2. User Stories / User Characteristics | 5  |
| 3. Functional Requirements             | 10 |
| 4. Service Contracts                   | 13 |
| 5. Domain Model                        | 18 |
| 6. Architectural Requirements          | 21 |
| 7. Technology Choices                  | 25 |
| 8. Technical Requirements              | 29 |

# 1. Introduction

The **FireFighter Access Management Platform** provides a structured, auditable solution for emergency privilege escalations, enabling rapid incident response with strict security controls.

## 1.1 Business Need

In BMW's complex IT environment, delays in granting elevated access during emergencies can lead to:

- Extended downtime in manufacturing operations
- Delayed resolution of critical security vulnerabilities
- Disruption to supply chain and logistics systems
- Operational inefficiencies and increased costs

## 1.2 Demo 1 Scope

In Demo 1 (28 May 2025), we present the front-end design with mocked data only. The demo covers polished UI flows for:

- **Login & Registration** (via Firebase Authentication)
- **Client-side Form Validation**
- **Theme Switching** (light/dark modes)
- **Dashboard Layout**
- **Notifications Page**
- **Ticket Requests Page**

## 1.3 Demo 2 Scope

In Demo 2 (27 June 2025), we present full-stack implementations for several key features that expand upon the front-end mock-ups from Demo 1. These include:

- **A fully functional landing page**
- **Full-stack ticket management system**
- **Real-time notification system**
- **Admin dashboard with session and role control**
- **Single Sign-On (SSO) integration with Firebase Auth and backend validation**

## 2. User Stories / User Characteristics

### 2.1 User Characteristics

#### 2.1.1. IT Engineer

- **Goal:** Resolve urgent system issues by obtaining temporary elevated access.
- **Tech Proficiency:** High; familiar with internal ticketing systems and command-line interfaces.
- **Pain Points:** Delays in access approval during emergencies; unclear access tracking.

#### 2.1.2. System Administrator

- **Goal:** Approve or reject FireFighter access requests and monitor sessions.
- **Tech Proficiency:** High; handles role assignments, logs, and compliance.
- **Pain Points:** Need for fast but secure access control, clear auditability, and minimal manual effort.

#### 2.1.3. Security Auditor

- **Goal:** Investigate incidents, verify system integrity, and analyse user activity.
- **Tech Proficiency:** Moderate to high; focuses on system logs and audit trails.
- **Pain Points:** Lack of centralized logging, ambiguity in change attribution.

#### 2.1.4. Manager / Incident Supervisor

- **Goal:** Be informed when emergency access is activated and ensure it is justified.
- **Tech Proficiency:** Medium; reads notifications, may not use CLI.
- **Pain Points:** Lack of visibility, excessive manual reporting.

## 2.2 User Stories

### 2.2.1 FireFighter Role Request Flow

1. **As an IT Engineer**, I want to request elevated access using a valid ticket ID so that I can fix critical issues during emergencies.
2. **As an IT Engineer**, I want the elevated role to automatically expire after a set time so that I do not risk forgetting to revoke it.
3. **As a System Administrator**, I want to receive real-time notifications when a FireFighter request is submitted so that I can respond quickly.

### 2.2.2 Approval and Governance

4. **As a System Administrator**, I want to approve or reject FireFighter requests based on predefined conditions so that access is controlled and auditable.
5. **As a Security Auditor**, I want to view a complete list of past FireFighter sessions and the actions performed so that I can ensure compliance.
6. **As a System Administrator**, I want to revoke access manually in critical cases so that I can maintain system integrity.

### 2.2.3 Notification & Monitoring

7. **As a Manager**, I want to receive a summary of FireFighter access events in my email or chat so that I stay informed about ongoing emergencies.
8. **As a User**, I want to get notified when my access is about to expire so that I can extend or wrap up my actions.

### 2.2.4 Auditing & Logging

9. **As a Security Auditor**, I want every change made during FireFighter access to be logged with user and timestamp so that actions can be traced.
10. **As a System Administrator**, I want a dashboard of ongoing FireFighter sessions and their statuses so that I can monitor live activity.

### 2.2.5 Landing Page Experience

11. **As a new user**, I want to understand what the platform does from the landing page so that I can decide whether to log in or register.
12. **As a returning user**, I want to be directed to the dashboard from the landing page for quick access to functionality.

### 2.2.6 Authentication & Account Management

13. **As a User**, I want to securely log in to the FireFighter system using my BMW credentials so that I can request emergency access when needed.

14. **As a User**, I want to register for a FireFighter account with my department information so that I can be authorized to submit emergency access requests.
15. **As a User**, I want to reset my password if I forget it so that I can regain access to the FireFighter system quickly during emergencies.
16. **As a User**, I want to view and update my account information including emergency contact details so that administrators can reach me during critical situations.

## 2.2.7 Emergency Access Requests

17. **As a User**, experiencing a system emergency, I want to quickly create an emergency access request by specifying the emergency type, affected system, justification, and required duration so that I can get timely access to resolve critical issues.
18. **As a User**, I want to select from predefined emergency types (Critical System Failure, Security Incident, Data Recovery, Network Outage, User Lockout, Other Emergency) so that my request is properly categorized for faster processing.
19. **As a User**, I want to set the duration of my emergency access (15 minutes to 2 hours) so that I receive appropriate time-limited access for the specific emergency.
20. **As a User**, I want to provide emergency contact information when submitting a request so that administrators can reach me directly if needed.
21. **As a User**, I want to view all my emergency access requests with their current status (Active, Completed, Rejected, Closed) so that I can track the progress of my requests.
22. **As a User**, I want to search and filter my requests by status and keywords so that I can quickly find specific emergency access requests.

## 2.2.8 Dashboard & Monitoring

23. **As a User**, I want to see a dashboard overview of my active tickets, total requests, and recent activity so that I can quickly understand my current emergency access status.
24. **As a User**, I want to view real-time statistics including approval rates, critical issues count, and success rates so that I can understand system performance and my request history.
25. **As a User**, I want to see recent activities and emergency requests from other users (anonymized) so that I can understand current system-wide emergency situations.
26. **As a User**, I want to see how much time is remaining on my active emergency access sessions so that I can plan my work accordingly.

## 2.2.8 Notifications

- 27. **As a User**, I want to receive real-time notifications when my emergency access request is approved, completed, or rejected so that I can take immediate action.
- 28. **As a User**, I want to view all my notifications in a centralized location so that I don't miss important updates about my emergency access requests.
- 29. **As a User**, I want to mark notifications as read and see unread notification counts so that I can manage my notification queue effectively.
- 30. **As a User**, I want to receive different types of notifications (ticket created, request completed, request approved, action taken) so that I'm informed of all relevant system events.

## 2.2.9 Administrative Functions

- 31. **As a System Administrator**, I want to view all active emergency access requests across the organization so that I can monitor ongoing emergency situations and system access.
- 32. **As a System Administrator**, I want to revoke emergency access for individual or multiple requests with a documented reason so that I can maintain security control over system access.
- 33. **As a System Administrator**, I want to view the complete history of all emergency access requests including completion status and audit logs so that I can maintain compliance and review access patterns.
- 34. **As a System Administrator**, I want to search and filter emergency requests by status, requester, system, and date range so that I can efficiently manage large volumes of access requests.
- 35. **As a System Administrator**, I want to export emergency access data to CSV format for audit reporting and compliance documentation so that I can fulfil regulatory requirements.
- 36. **As a System Administrator**, I want to see detailed information about each request including justification, emergency contact, affected systems, and requester details so that I can make informed decisions about access control.

## 2.2.10 Security & Audit

- 37. **As a Security Auditor**, I want all emergency access activities to be automatically logged and audited so that I can ensure compliance with security policies.

- 38. **As a Security Auditor**, I want to see failed login attempts and unauthorized access attempts so that I can investigate potential security breaches.
- 39. **As a Security Auditor**, I want to generate comprehensive audit reports showing all emergency access grants, revocations, and usage patterns so that I can demonstrate compliance with regulatory requirements.

## 2.2.11 Mobile & Accessibility

- 40. **As a User** working in the field, I want to access the FireFighter system from my mobile device so that I can request emergency access even when I'm not at my desktop.
- 41. **As a User**, I want the system to work seamlessly in both light and dark modes so that I can use it comfortably in different lighting conditions.
- 42. **As a User**, I want to use pull-to-refresh functionality on mobile to update my request status so that I can get the latest information without navigating through menus.

## 2.2.12 System Integration

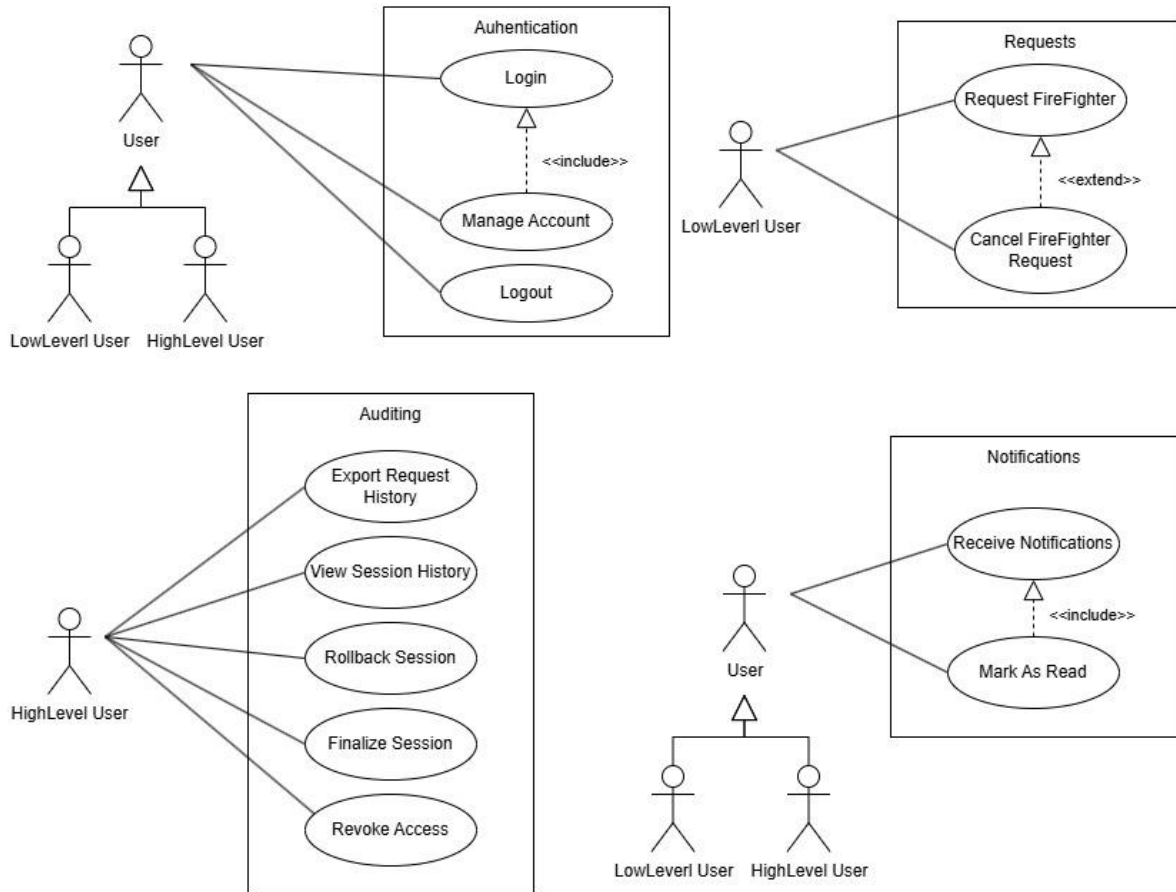
- 43. **As a User**, I want the system to integrate with BMW's existing authentication infrastructure so that I can use my standard BMW credentials without creating separate accounts.
- 44. **As a User**, I want the system to automatically map emergency types to affected BMW systems (BMW-SAP-ERP, BMW-MES-PROD, BMW-PLM-SYS, etc.) so that administrators understand exactly which systems require emergency access.

## 2.2.13 Help & Support

- 45. **As a User**, I want access to comprehensive help documentation and FAQs so that I can understand how to properly use the FireFighter system during emergencies.
- 46. **As a User**, I want to know the system's emergency contact information and escalation procedures so that I can get help when the FireFighter system itself is unavailable.
- 47. **As a User**, I want to understand the emergency access duration limits and extension procedures so that I can plan my emergency response activities effectively.



## 2.3. Use Case Diagram



## 3. Functional Requirements

### 1.1. User Authentication and Registration

**1.1.1.** The system will allow users to register for an account using a secure, email-based sign-up process.

**1.1.2.** The system will allow registered users to log in securely using their credentials.

**1.1.3.** The system will maintain user session state across both web and mobile interfaces.

**1.1.4.** The system will enforce email verification or SSO before granting access to sensitive features.

### 1.2. Role Request Management

**1.2.1.** Allow authorized users to make/send a request for temporary FireFighter access by entering a valid ticket ID.

**1.2.2.** The system shall verify that the requesting user is authorized to request extended access.

**1.2.3.** The ticket ID used in a request will be validated before access is granted.

**1.2.4.** The system will notify the user whether the access they requested was approved or denied.

### 1.3. Time-Bound access control

**1.3.1.** The system will grant FireFighter access for a set duration as specified in the application settings.

**1.3.2.** The system shall remove the FireFighter access once the access duration expires.

**1.3.3.** The system will allow administrators to manually revoke active FireFighter roles before the expiry time.

**1.3.4.** The systems will allow users to terminate their own access early.

## 1.4. Action Attribution and Logging

**1.4.1.** The system will record all actions performed under an active FireFighter role session.

**1.4.2.** For each action performed, the system will store the user responsible, the ticketID associated with the session, and the timestamp of the action.

**1.4.3.** The system will ensure that logs are stored securely so that the logs cannot be tampered with.

## 1.5. Notification system

**1.5.1.** The system shall send a notification to administrators when a FireFighter role request is submitted.

**1.5.2.** The system will notify administrators when a FireFighter role is granted and becomes active.

**1.5.3.** The system will notify administrators when a FireFighter role is revoked.

**1.5.4.** The system will provide confirmation notifications to users during request, activation, and revocation.

## 1.6. Security and Access control

**1.6.1.** The system will enforce role-based access control to restrict access to sensitive operations.

**1.6.2.** The system shall authenticate all users using Firebase.

**1.6.3.** The system will log all failed access attempts, providing relevant context for security auditing.

**1.6.4.** The system will assign administrator roles through secure configuration.

**1.6.5.** Only users with the administrator role will access administrative features such as role revocation, session monitoring, and audit logs.

## 1.7. Admin Dashboard

**1.7.1.** The system will provide an interface for administrators to view all pending, active, and expired FireFighter sessions.

**1.7.2.** The system will allow filtering and searching through historical logs for auditing.

## 1.8. Ticketing System Integration

**1.8.1.** The system will support integration with an external ticketing system to retrieve and validate ticket details.

**1.8.2.** The system will fall back to a mock ticketing system if no external system is integrated.

## 1.9. Chatbot Interface

**1.9.1.** The system will provide a chatbot interface which will allow users to request and manage FireFighter roles via natural language.

**1.9.2.** The chatbot will support access through platforms like a web-based chat, Microsoft Teams, or Slack.

**1.9.3.** The chatbot will guide users through the request process using contextual prompts.

## 1.10. Landing Page

**1.10.1.** The system shall serve a dedicated landing page as the public entry point to the application.

**1.10.2.** The landing page will provide information about the platform, login/register buttons, and user guidance.

## 4. Service Contracts

This section defines the service contracts for the FireFighter Access Management Platform, focusing on interaction protocols between the **Angular/Ionic front-end**, **Spring Boot back-end**, **Firebase Authentication** and **PostgreSQL DB**. It ensures consistency, security, and traceability across all components.

### 4.1. Authentication Service (Firebase OAuth)

**Provider:** Firebase Authentication

**Used By:** Angular/Ionic Frontend

**Purpose:** Secure sign-in via SSO/OAuth2 and ID token retrieval.

#### Endpoint Behaviour

| Feature              | Details                                    |
|----------------------|--|
| <b>Auth Provider</b> | Firebase OAuth2                            |
| <b>Token Format</b>  | JWT (Firebase ID Token)                    |
| <b>Header</b>        | Authorization: Bearer <ID_TOKEN>           |
| <b>Expiry</b>        | Set by Firebase (usually 1 hour)           |
| <b>Verification</b>  | Handled server-side via Firebase Admin SDK |

- **POST /api/users/verify**

Content-Type: application/x-www-form-urlencoded

Request Params: firebaseUid, username, email, department (optional)

Response: User object (with roles, admin status, etc.)

## 4.2. Access Request Service

**Endpoint:** POST /api/tickets

**Description:** Allows authorised users to request FireFighter access.

**Authorization:** The frontend is responsible for ensuring the user is authenticated; the backend expects user info in the request.

### Request JSON

```
{
  "ticketId": "INC123456",
  "description": "Urgent patch on DB cluster",
  "userId": "user@example.com",
  "emergencyType": "critical-system-failure",
  "emergencyContact": "+49 123 456789",
  "duration": 30
}
```

### Response JSON

```
{
  "id": 1,
  "ticketId": "INC123456",
  "description": "Urgent patch on DB cluster",
  "status": "Active",
  "dateCreated": "2024-05-26T12:00:00Z",
  "userId": "user@example.com",
  "emergencyType": "critical-system-failure",
  "emergencyContact": "+49 123 456789",
  "duration": 30
}
```

## Error Handling

| Cod<br>e | Reason                         |
|----------|--------------------------------|
| 400      | Missing or invalid fields      |
| 401      | No/invalid token               |
| 403      | User not authorised to request |
| 409      | Duplicate active request       |
| 500      | Server error                   |

### 4.3. Role Activation & Expiry

**Endpoint:** POST /api/users/{firebaseUid}/roles

**Effect:** Assign role

**Endpoint:** PUT /api/access-request/{firebaseUid}/revoke

**Effect:** Revoke authorization

**Endpoint:** PUT /api/access-request/{firebaseUid}/authorize

**Effect:** Authorization

#### Automatic Revocation

- Role revoked automatically after duration
- Manual PUT /api/access-request/{firebaseUid}/revoke for early deactivation

### 4.4. Audit Logging Service

**Tech:** Data is stored in the access\_logs table in PostgreSQL

#### Data Stored:

- Who requested access
- Ticket ID
- Access time window
- Actions performed

#### Format (Sample Log Document)

```
{
  "userId": "user@example.com",
  "ticketId": "INC123456",
  "action": "APPROVE_ACCESS",
  "timestamp": "2025-05-26T10:12:00Z",
  "sessionId": 123,
}
```

## 4.5. Notification Service

### Triggers:

- New ticket created
- Ticket status changes
- Role expired or manually revoked

**Delivery:** Handled by frontend with an in-app notification service

### Example Payload:

```
{
  "type": "ticket_created",
  "title": "New Ticket Created",
  "message": "A new ticket INC123456 has been created.",
  "ticketId": "INC123456"
}
```

## 4.6. Ticket Validation

**Endpoint:** GET /api/tickets/ticket-id/{ticketId}

### Response:

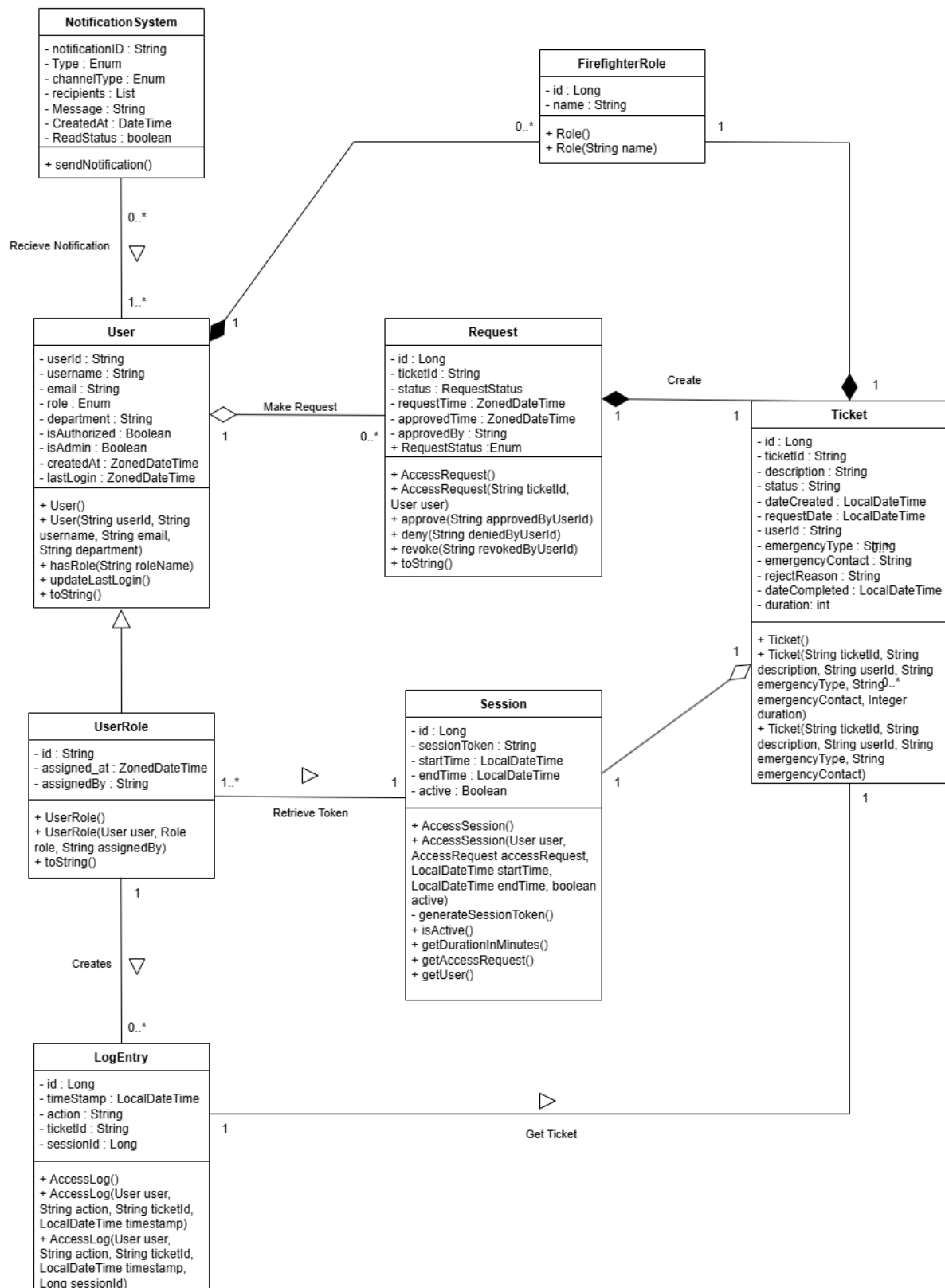
```
{
  "ticketId": "INC123456",
  "description": "Urgent patch on DB cluster",
  "status": "Active",
  "userId": "user@example.com",
  "emergencyType": "critical-system-failure",
  "emergencyContact": "+49 123 456789",
  "duration": 30
}
```



## 4.7. Technology Contract Summary

| Component               | Interface Type  | Protocol    | Notes                                 |
|-------------------------|-----------------|-------------|---------------------------------------|
| <b>Frontend (Ionic)</b> | RESTful API     | HTTPS       | Calls Spring Boot API via NGINX proxy |
| <b>Backend (Spring)</b> | REST Controller | HTTPS + JWT | Verifies Firebase tokens              |
| <b>DB (PostgreSQL)</b>  | JDBC / ORM      | Localhost   | Hibernate ORM                         |
| <b>Logs</b>             | DB Table        | -           | Stored in access_logs table.          |
| <b>CI/CD</b>            | GitHub Actions  | YAML        | Auto-build/test/deploy containers     |

## 5. Domain Model



This figure illustrates key domain entities and their relationships in the FireFighter platform:

## User & Administrator

- A **User** is an individual who can initiate access requests for elevated privileges during emergencies.
- An **Administrator** is responsible for approving, denying, or revoking these access requests.
- Users are associated with one or more **UserRoles**, defining their default or emergency permissions.
- The system supports **Single Sign-On (SSO)**, and authentication sessions are managed via **Session** entities which include security tokens.

## Ticket & TicketingSystem

- Every emergency access request results in the creation of a **Ticket**.
- Tickets track the lifecycle of a request, from submission to closure, and include metadata such as reason, requested scope, and timestamps.
- The **TicketingSystem** automates the creation, status updates, and resolution tracking of tickets.
- Once a Ticket is approved, it triggers provisioning of access rights via **FireFighterRole** assignment.

## FireFighterRole & Session

- A **FireFighterRole** is a predefined role granting elevated access for a limited duration.
- Upon ticket approval, the User is temporarily assigned this role, and a **Session** is created representing this active access state.
- The Session has:
  - A time-limited validity period
  - A unique token for secure identification
  - Links to associated SystemActions for traceability
- If the access is revoked or expires, the Session is terminated.

## SystemAction & SecurityValidation

- Any privileged operation performed by a User during an active Session is tracked as a **SystemAction**.
- Each SystemAction is checked against predefined security rules through **SecurityValidation**, which:
  - Ensures actions align with policies and do not exceed the intended scope
  - Logs violations or anomalies
- This validation process is essential for compliance and auditing.

## NotificationSystem

- The **NotificationSystem** handles real-time updates for key events:
  - Access Request initiation
  - Approval or denial by Admin
  - Session activation or termination
  - Role revocation
- Notifications are delivered to relevant stakeholders via preferred channels (email, dashboards, etc.).
- Users and administrators rely on this system for situational awareness and quick response.

## LogEntry

- Every significant event (such as Ticket creation, Session start/end, Role assignment, or SystemAction) is recorded as a **LogEntry**.
- These logs serve:
  - Forensically, in case of an incident
  - Operationally, for monitoring
  - Audit-wise, for regulatory or internal compliance

## UserRole

- Represents the permissions or access levels a User holds, either permanently or temporarily.
- UserRoles are evaluated during the Session and influence what SystemActions are permissible.
- Role-to-Session mappings are critical for maintaining least-privilege access.

## Session Management & Token Retrieval

- The **Session** object includes a security token retrieved during login or SSO.
- This token is required for all subsequent operations and is validated before any access is granted.
- It ensures secure and traceable access to sensitive resources.

## 6. Architectural Requirements

### 6.1 Architectural Design Strategy

We adopted a **quality-driven design approach**, as our system's primary concerns are **security** and **scalability**.

Given BMW's enterprise scale and volume of sensitive data (trademarks, financials, PII), security and scalability are non-negotiable core qualities.

A quality-driven approach ensures that our architectural choices are explicitly designed to satisfy critical non-functional requirements. It allows us to:

- Prioritize architectural decisions that enforce **role-based access control**, data protection, and auditability.
- Design for **scalability** so the system can operate effectively on both modest internal hardware and large-scale AWS deployments.
- Ensure **availability and responsiveness** in a security-first context.

### 6.2 Architectural Style

3-Tier Layered Modular Monolith

- **Frontend:** Ionic + Angular web app (with Capacitor for mobile)
- **Middle tier (API):** Spring Boot
- **Persistence tier:** PostgreSQL

**Security:** The API layer (Spring Boot) acts as a gateway, enforcing access controls and preventing direct client access to the database. This layered approach reduces attack surface.

**Clear separation of concerns:** Each layer has a distinct responsibility (UI, business logic, data), simplifying development, testing, and maintenance.

**Spring Boot modular monolith benefits:**

- Combines the maintainability of modular design with the deployment simplicity of a monolith.
- Provides built-in support for security (Spring Security), validation, transaction management, and REST API design.

**Operational Simplicity:** Single deployable artifact reduces cloud deployment overhead while maintaining scalability via AWS autoscaling.

## 6.3 Architectural Quality Requirements

### 1. Security

All API endpoints protected by role-based access control. Admin-only actions (e.g. session revocation, data export) require an admin role. All data encrypted in transit

### 2. Scalability

System configurable to support at least 500 concurrent requests/sec in AWS staging, adjustable with provisioned resources.

### 3. Availability

System must maintain >99% uptime over any 30-day period, with planned maintenance windows announced in advance.

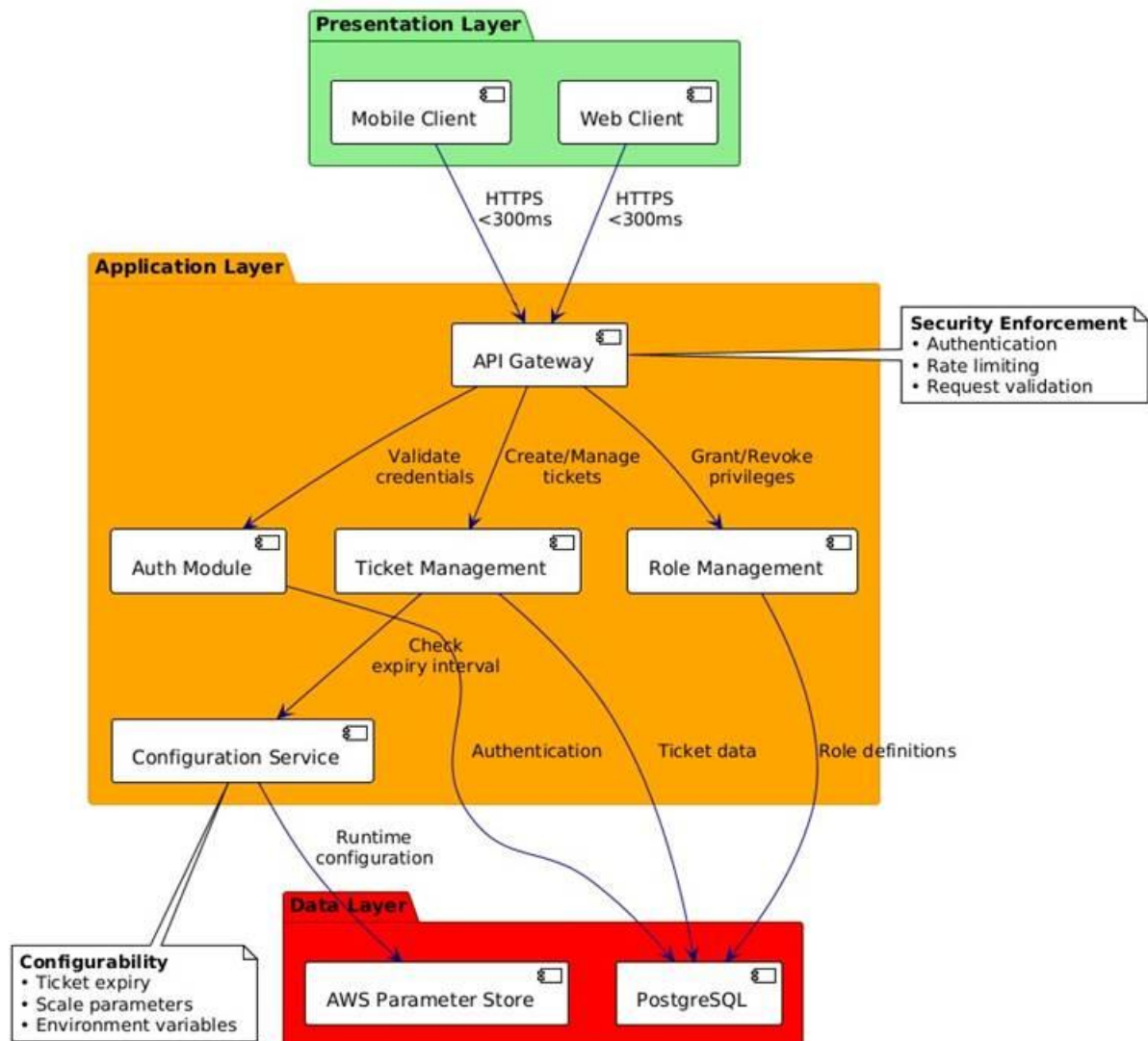
### 4. Responsiveness

Mobile app must respond to user interactions within <300ms on modern devices; backend endpoints must respond within deployment-specified thresholds (e.g. <500ms for core API calls under normal load).

### 5. Interoperability

The system must expose API endpoints or provide export formats compatible with existing business systems (e.g. CSV exports for audit systems).

## 6.4 Architectural Design and Pattern





## 6.5 Architectural Constraints

- **Open-source requirement:**  
All components must be based on open-source technologies to avoid licensing costs and vendor lock-in.
- **Deployment requirement:**  
System must be containerized and deployable on AWS infrastructure using Docker.
- **Closed ecosystem:**  
System is not public-facing; only employees and administrators within the corporate environment will have access.

## 7. Chosen Technologies

### Frontend

#### Prospective technologies

| Technology                         | Pros   | Con   |
|------------------------------------|--|---|
| <b>React Native</b>                | <ul style="list-style-type: none"> <li>· Strongly encourages code reuse in both web &amp; mobile applications</li> <li>· Large ecosystem</li> </ul>                        | <ul style="list-style-type: none"> <li>· Requires much configuration for web-mobile parity</li> <li>· Frequent plugin conflicts</li> </ul>                                    |
| <b>Flutter</b>                     | <ul style="list-style-type: none"> <li>· Excellent performance and...</li> <li>· Conducive to an all-around good mobile experience</li> </ul>                              | <ul style="list-style-type: none"> <li>· Learning curve. Dev team has never used Dart</li> <li>· Unexemplary web support</li> </ul>   |
| <b>Ionic + Angular + Capacitor</b> | <ul style="list-style-type: none"> <li>· Shared codebase between web &amp; mobile</li> <li>· Rich ecosystem</li> <li>· Capacitor enables native device features</li> </ul> | <ul style="list-style-type: none"> <li>· Large bundle size (multiple installations and employed frameworks)</li> <li>· Learning curve for uninitiated team members</li> </ul> |

#### Final decision:

Ionic + Angular + Capacitor

- Inherently fits planned layered architecture.
- Code reuse is invaluable for time constrained projects
- Some team members have previous experience

## Backend

### Prospective technologies

| Technology         | Pros   | Cons   |
|--------------------|--|--|
| <b>Node.js</b>     | <ul style="list-style-type: none"> <li>· Lightweight</li> <li>· Easy to get started</li> </ul>   | <ul style="list-style-type: none"> <li>· Not feature rich.</li> <li>· Requires additional setup for security and module patterns</li> </ul>                        |
| <b>Django</b>      | Exceptionally feature rich   | <ul style="list-style-type: none"> <li>· Incredibly slow Python runtime environment</li> <li>· Modularity comparable to (if not worse than) Spring Boot</li> </ul> |
| <b>Spring Boot</b> | <ul style="list-style-type: none"> <li>· Built in security</li> <li>· Excellent REST API support</li> <li>· Encourages modularity</li> </ul> | Heavier runtime cost   |

### Final decision:

#### Spring Boot

- API and modularity align with intended layered architecture.
- Inherent security features
- Modularity

## Database

### Prospective technologies

| Technology        | Pros  | Cons                                     |
|-------------------|---|--|
| <b>PostgreSQL</b> | Open source<br>Wide AWS support<br>Powerful | More complex setup compared to noSQL DBs |
| <b>MySQL</b>      | Mature<br>Wide support                      | Sparse features                          |
| <b>MongoDB</b>    | Flexible<br>Scalable                        | Not ideal for relational data            |

### Final decision:

PostgreSQL

- Strong support for relational data integrity
- Inherent support for containerized deployment

## Dependency Management

### Prospective technologies

| Technology    | Pros   | Cons  |
|---------------|--|---|
| <b>Maven</b>  | <ul style="list-style-type: none"> <li>· Stable</li> <li>· Widely used in Java development</li> <li>· Frequently used in conjunction with Spring Boot</li> </ul> | Verbose (easy human interaction)  |
| <b>Gradle</b> | <ul style="list-style-type: none"> <li>· Flexible</li> <li>· Fast project building</li> </ul>  | <ul style="list-style-type: none"> <li>· Complicated configuration</li> <li>· Learning Curve</li> </ul> |
| <b>Ant</b>    | Customizable   | Outdated  |

Final decision:

Maven

- Entire team has experience with its use
- Easy deployment
- Frequently used in conjunction with Spring Boot
- Easy integration with Docker

## 8. Technical Requirements

This section specifies the environments, version constraints, performance/security targets, data-management policies, reliability, scalability, monitoring, and CI/CD workflow for Demo 1 and beyond.

(All of the following requirements are subject to change and are not final.)

### 8.1. External Interface Requirements

- **Desktop Browsers:** Chrome, Firefox, Edge (latest two major versions)
- **Mobile Browsers:** Safari on iOS 14+, Chrome on Android 11+
- **Development Hosts:** Windows, macOS, Linux (for local builds and CI runners)

### 8.2. Design Constraints

- **Node.js & npm:**  $\geq 16.x$  /  $\text{npm} \geq 8.x$
- **Angular & Ionic Capacitor:** Angular LTS; Ionic Capacitor 4.x
- **Java & Spring Boot:** OpenJDK 11+; Spring Boot 2.6+
- **Database:** PostgreSQL v13; Hibernate ORM 5.6+
- **Containerisation:** Docker 20.10+ (for local dev and future services)
- **CI Runner:** GitHub Actions (ubuntu-latest)

### 8.3. Deployment & Infrastructure

- **Hosting:** AWS Free-Tier EC2 (e.g., t2.micro) instances for all application tiers
- **Backup Strategy:** Weekly AMI snapshots of EC2 instances

### 8.4. Security & Compliance

- **Authentication & User Management:** Firebase Authentication (OAuth/SSO via Google, Microsoft, etc.) with built-in user directory and token handling
- **Authorization:** Role-based rules enforced client-side and via Spring Security guards
- **Transport Security:** TLS 1.2+ for all endpoints
- **OWASP Top 10:** CSP headers, XSS/CSRF protections in Angular, secure cookie flags

## 8.5. Performance Requirements

- **First-Paint:**  $\leq 2$  s on a 4G connection
- **UI Responsiveness:** Interactive actions (theme switch, form validation)  $\leq 200$  ms
- **Mock Data Fetch:**  $\leq 100$  ms round-trip for local API stubs
- **Capacity Target:** Architected for  $\geq 1\,000$  concurrent users in future back-end demos

## 8.6. Data Management

- **Demo 1:** In-memory JSON fixtures only
- **Future Production:**
  - **Backups:** Nightly PostgreSQL dump to S3
  - **Retention:**
    - **Database:** 30 days
    - **Audit Logs:** 90 days
  - **Encryption:** AES-256 at rest (via RDS or disk-level encryption)

## 8.7. Reliability & Availability

- **Uptime Target:** 99.9% over any 30-day period (post-Demo 1)
- **Health Checks:** Automated smoke tests of critical UI routes and mock services
- **Error Handling:** Graceful fallback screens with “retry” options

## 8.8. Scalability & Capacity Planning

- **Stateless Front-end:** Served from EC2; horizontal scaling by adding EC2 instances behind an ELB
- **Session Management:** Firebase tokens stored client-side; no server affinity required

## 8.9. Monitoring & Logging

- **Centralised Logging:** ELK Stack on EC2 (Elasticsearch, Logstash, Kibana)
- **Metrics & Alerts:**
  - **Error Rate:** Alert if > 1% of UI/API calls fail over 5 min
  - **Latency:** Alert if average API response > 500 ms over 10 min
- **Audit Trail:** Immutable Firebase “last sign-in” logs plus application activity logs

## 8.10. CI/CD & Quality Gates

- **Pipeline Stages:**
  1. Build & unit tests (Angular CLI; JUnit/Mockito)
  2. Static analysis (ESLint, Checkstyle), security scans (Snyk)
  3. End-to-end tests (Cypress on mock data)
  4. Docker image build & push to AWS ECR
- **Quality Gates:**
  - **Coverage:**  $\geq 80\%$
  - **Linting:** Zero errors
  - **Vulnerabilities:** No high/critical findings

## 8.11. Compliance & Standards

- **Data Protection:** GDPR compliance for user data
- **Security Framework:** ISO/IEC 27001 alignment for access management
- **Documentation:**
  - **API:** Swagger UI (OpenAPI v3)
  - **Runbooks:** Versioned Markdown for deployment, rollback, and recovery