SOFTWARE REQUIREMENTS SPECIFICATION

BMW IT HUB

COS 301 | University of Pretoria

FIRE-FIGHTER ACCESS MANAGEMENT PLATFORM

Contents

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 3 Scope

For demo 3 (20 August 2025), we focused on refining existing systems and improving the user experience:

- Implemented JWT tokens for enhanced security
- Implemented Gemeni as an AI assistant to query tickets using natural language
- Implemented a mailing service to notify the user when they are not actively using the application
- Implemented a mock metrics page to showcase upcoming analytics functionality
- Minor improvements to the user interface and experience

2. User Stories / User Characteristics

2.1 User Characteristics

2.1.1. Software Engineer (High Level User)

- **Goal**: Resolve urgent system issues by obtaining temporary elevated access.
- **Tech Proficiency**: High Level understanding of systems. Familiar with internal ticketing systems.
- **Pain Points**: Delays in access approval during emergencies; unclear access tracking.

2.1.4. IT-HUB Employee (Low Level User)

- **Goal**: Make quick bug fixes to system by obtaining temporary elevated access
- **Tech Proficiency**: Low Level understanding of systems.
- **Pain Points**: Potential misuse of Elevated privileges, whether intentional or not.

2.1.2. System Administrator / Manager

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

2.2 User Stories

2.2.1 FireFighter Role Request Flow

1. **As a Software Engineer**, I want to request elevated access using a valid ticket ID so that I can fix critical issues during emergencies.
2. **As a Software 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.

2.2.4 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 Software Engineer**, 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 an IT-HUB Employee**, 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 Software Engineer,** 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 an IT-HUB Employee**, 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 Software Engineer,** 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 an IT-HUB Employee**, 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 Software Engineer**, 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 an IT-HUB Employee**, I want to provide emergency contact information when submitting a request so that administrators can reach me directly if needed.
21. **As a Software Engineer**, 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 Software Engineer**, 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 Software Engineer**, 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 Manager**, I want to receive a summary of FireFighter access events in my email or chat so that I stay informed about ongoing emergencies.
25. **As a Software Engineer**, 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.
26. **As an IT-HUB Employee**, I want to see recent activities and emergency requests from other users (anonymized) so that I can understand current system-wide emergency situations.
27. **As a Software Engineer**, 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

28. **As a Software Engineer**, I want to receive real-time notifications when my emergency access request is approved, completed, or rejected so that I can take immediate action.
29. **As an IT-HUB Employee**, I want to get notified when my access is about to expire so that I can extend or wrap up my actions.
30. **As a Software Engineer**, I want to view all my notifications in a centralized location so that I don't miss important updates about my emergency access requests.
31. **As an IT-HUB Employee**, I want to mark notifications as read and see unread notification counts so that I can manage my notification queue effectively.
32. **As a Software Engineer**, 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

33. **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.
34. **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.
35. **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.
36. **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.
37. **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.
38. **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

39. **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.
40. **As a Security Auditor**, I want to see failed login attempts and unauthorized access attempts so that I can investigate potential security breaches.
41. **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

42. **As a Software Engineer, -**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.
43. **As an IT-HUB Employee**, I want the system to work seamlessly in both light and dark modes so that I can use it comfortably in different lighting conditions.
44. **As a Software Engineer**, 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

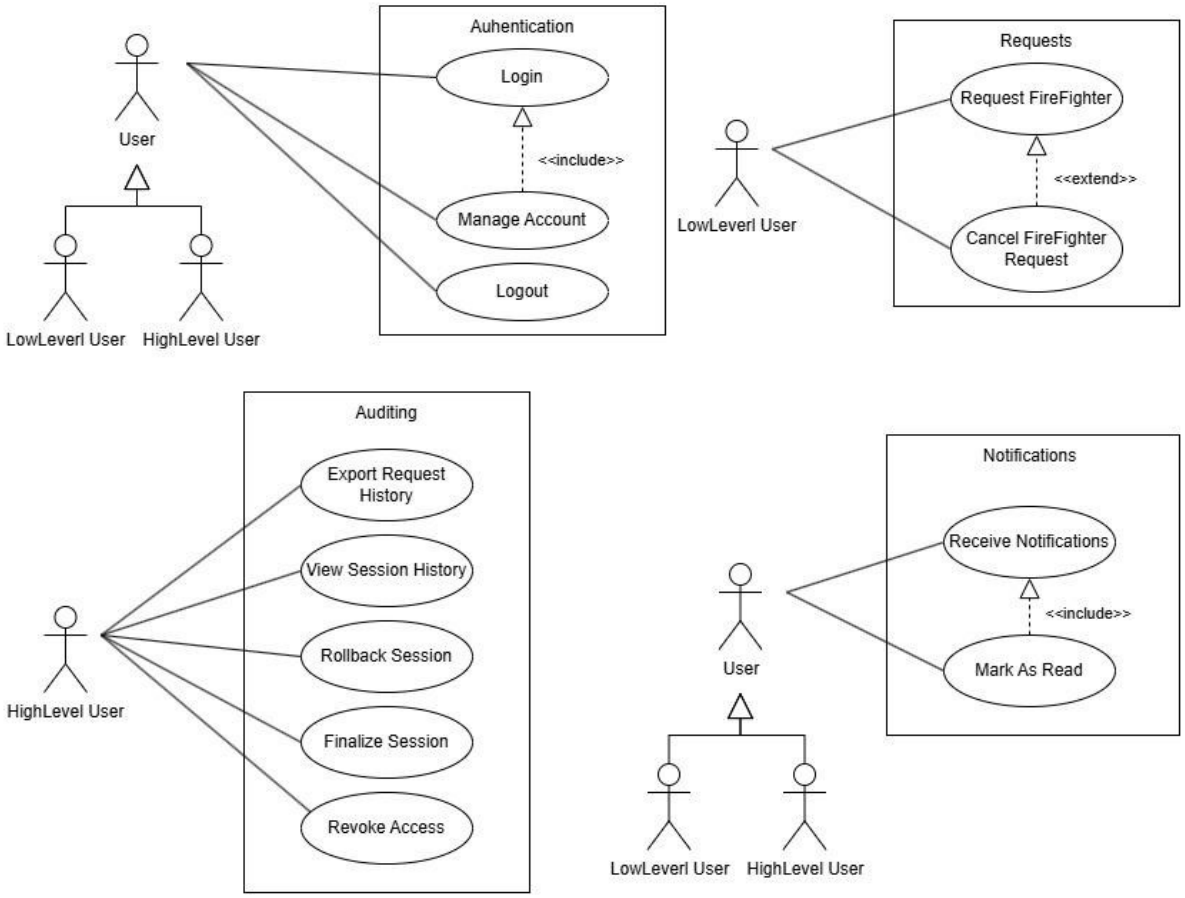
45. **As a Software Engineer**, I want the system to integrate with BMW's existing authentication infrastructure so that I can use my standard BMW credentials without creating separate accounts.
46. **As a System Administrator**, I want the system to automatically map emergency types to affected BMW 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

47. **As an IT-HUB Employee**, I want access to comprehensive help documentation and FAQs so that I can understand how to properly use the FireFighter system during emergencies.
48. **As a Software Engineer**, 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.

49. **As an IT-HUB Employee**, 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 shall allow users to register for an account using a secure, email-based sign-up process.

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

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

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

1.2. Role Request Management

1.2.1. The system shall allow authorized users to request 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 shall be validated before access is granted.

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

1.3. Time-Bound Access Control

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

1.3.2. The system shall automatically revoke FireFighter access once the duration expires.

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

1.3.4. The system shall allow users to terminate their own access early.

1.4. Action Attribution and Logging

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

1.4.2. For each action performed, the system shall store the user responsible, the ticket ID associated with the session, and the timestamp of the action.

1.4.3. The system shall store logs in a secure, tamper-resistant format.

1.4.4. The system shall maintain an immutable audit trail for all security-related events (logins, failed authentication attempts, token usage, API key lifecycle events, role grants/revocations).

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 shall notify administrators when a FireFighter role is granted and becomes active.

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

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

1.5.5. The system shall support security alerts (e.g., unusual login activity, abnormal API key usage).

1.5.6. The system shall support a five-minute expiry warning by creating an in-app notification and, if enabled by user preferences, send an email warning

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.

1.6.6. The system shall support API key–based authentication for service-to-service communication, with the ability to activate, deactivate, and revoke keys.

1.6.7. The system shall support JWT-based stateless authentication with tokens containing user claims, including firebaseUid, email, and isAdmin.

1.7. Admin Dashboard

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

1.7.2. The system shall allow administrators to filter and search through historical logs for auditing.

1.7.3. The system shall provide administrators with tools to manage API keys and monitor their usage.

1.7.4. The system shall provide an admin-only metrics page displaying real-time system performance and usage statistics accessible only through role-based authentication.

1.8. Ticketing System Integration

1.8.1. The system shall integrate with an external ticketing system to retrieve and validate ticket details.

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

1.9. Chatbot Interface

1.9.1. The system shall provide a chatbot interface that allows users to request and manage FireFighter roles via natural language.

1.9.2. The chatbot shall 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 shall provide information about the platform, login/register options, and user guidance.

1.11. User Profile Page

1.11.1. The system shall allow users to save user preferences and contact information (phone number).

1.11.**2.** The system shall provide user notification preferences allowing users to enable/disable email notifications for ticket creation, completion, revocation, and five-minute warnings.

1.11.3. The system shall allow users to reset their notification preferences to default values.

1.12. User Support and Documentation

1.12.1. The system shall provide in depth user documentation that can be accessed through the web interface.

1.12.2. The system shall include contextual help and tips for some features such as API key management and notification preferences.

1.12.3. The system shall include API documentation (Swagger/OpenAPI) for developers integrating with the platform.

1.12.4. The system shall provide troubleshooting guides for common issues such as authentication failures and notification delivery problems.

1.12.5. The system shall include a chatbot help system that can answer frequently asked questions about platform usage.

1.12.6. The system shall provide step-by-step guides for common workflows such as requesting FireFighter access and managing API keys.

1.12.7. The system shall maintain up-to-date documentation for all API endpoints with examples and response formats.

1.12.8. The system shall provide security best practices documentation for API key management and session handling.

1.12.9. The system shall provide a changelog listing the updates that come with each version of the application.

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
Auth Provider	Firebase OAuth2
Token Format	JWT (Firebase ID Token)
Header	Authorization: Bearer <ID_TOKEN>
Expiry	Set by Firebase (usually 1 hour)
Verification	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

Code	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

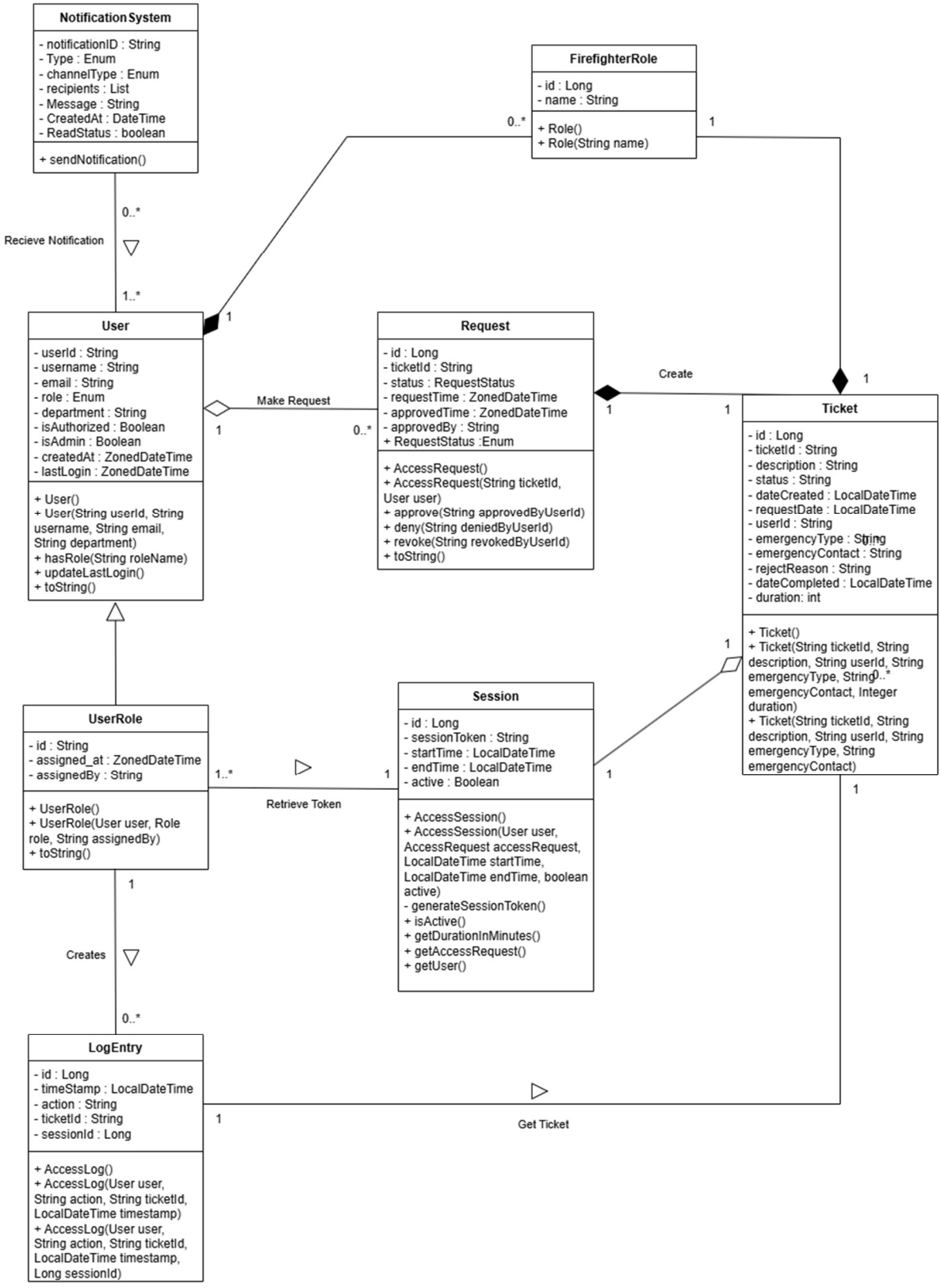
Endpoint: GET /api/tickets/ticket-id/{ticketId}
Response:

```
{
  "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
Frontend (Ionic)	RESTful API	HTTPS	Calls Spring Boot API via NGINX proxy
Backend (Spring)	REST Controller	HTTPS + JWT	Verifies Firebase tokens
DB (PostgreSQL)	JDBC / ORM	Localhost	Hibernate ORM
Logs	DB Table	-	Stored in access_logs table.
CI/CD	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 result 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

Client-Server Architecture:

- **Client:** Ionic + Angular web app (with Capacitor for mobile)
  - Lightweight and easily portable user interface for interacting with the system by sending service requests
- **Server:** Spring Boot API + PostgreSQL Database
  - All major requests are processed on a remote server.
  - Ensures control over system environment
  - Improves security by ensuring the system can only be interacted with through specific endpoints (reduced attack surface)
  - Hides system underpinnings

3-Tier Architecture:

- **Client, Server, External Services:**
  - The client user interfaces are local installations of the mobile app on the client smartphone, and the web app accessible via a desktop browser.

- The server tier encapsulates the BMW Fire-Fighter server software and its database deployed together on the same machine/container
- The Fire-Fighter system is inherently designed to integrate with external business systems in order to grant a Fire-Fighter client elevated privileges on said business systems.
  The Fire-Fighter system also facilitates an AI-assistant service which is hosted externally.
  This tier is designed to allow the services within it to be modularly exchangeable.

Model **V**iew **C**ontroller:

- **V**iew: A client may be presented with system data via appropriate and interchangeable representations
- **C**ontroller: Data access and presentation are handled by API endpoints.
- **M**odel: The fundamental representation of the system and data upon which it operates

Layered Architecture:

- The Server tier is logically separated into layers wherein each provides utility for the layer above it.
- This improves component autonomy and ease of system modification/maintenance.

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 the 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.
   - Estimated Mean Time to Failure: 340 hours
   - Estimated Hotfix Time: 3 hours
     Reboot Time: 5 Minutes
   - Estimated Availability = 340 hrs / ((3hrs+5 Minutes) +340hrs) = 99.1%

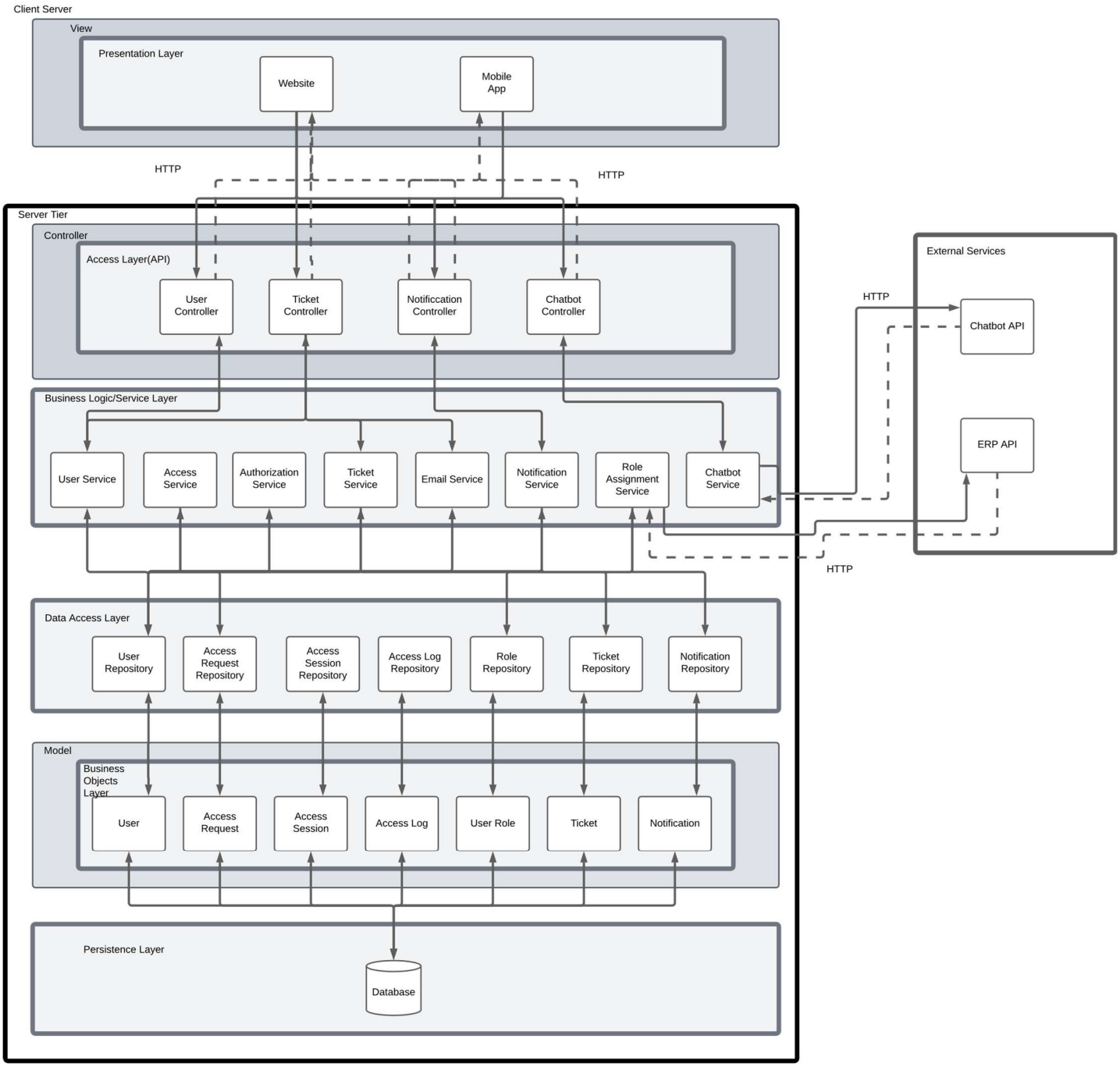
4. **Responsiveness**

   - 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 Patterns



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
React Native	· Strongly encourages code reuse in both web & mobile applications · Large ecosystem	· Requires much configuration for web-mobile parity · Frequent plugin conflicts
Flutter	· Excellent performance and... · Conducive to an all-around good mobile experience	· Learning curve. Dev team has never used Dart · Unexemplary web support
Angular + Capacitor	· Shared codebase between web & mobile · Rich ecosystem · Capacitor enables native device features	· Large bundle size (multiple installations and employed frameworks) · Learning curve for uninitiated team members

Final decision:

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
Node.js	· Lightweight · Easy to get started	· Not feature rich. · Requires additional setup for security and module patterns
Django	Exceptionally feature rich	· Incredibly slow Python runtime environment · Modularity comparable to (if not worse than) Spring Boot
Spring Boot	· Built in security · Excellent REST API support · Encourages modularity	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
PostgreSQL	Open source Wide AWS support Powerful	More complex setup compared to NoSQL DBs
MySQL	Mature Wide support	Sparse features
MongoDB	Flexible 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
Maven	· Stable · Widely used in Java development · Frequently used in conjunction with Spring Boot	Verbose (easy human interaction)
Gradle	· Flexible · Fast project building	· Complicated configuration · Learning Curve
Ant	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:** ≥ 16.x / npm ≥ 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:** ≤ 2 s on a 4G connection
- **UI Responsiveness:** Interactive actions (theme switch, form validation) ≤ 200 ms
- **Mock Data Fetch:** ≤ 100 ms round-trip for local API stubs
- **Capacity Target:** Architected for ≥ 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:** ≥ 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

9. Deployment Model

9.1. Target Environment

Hybrid Deployment Model combining:

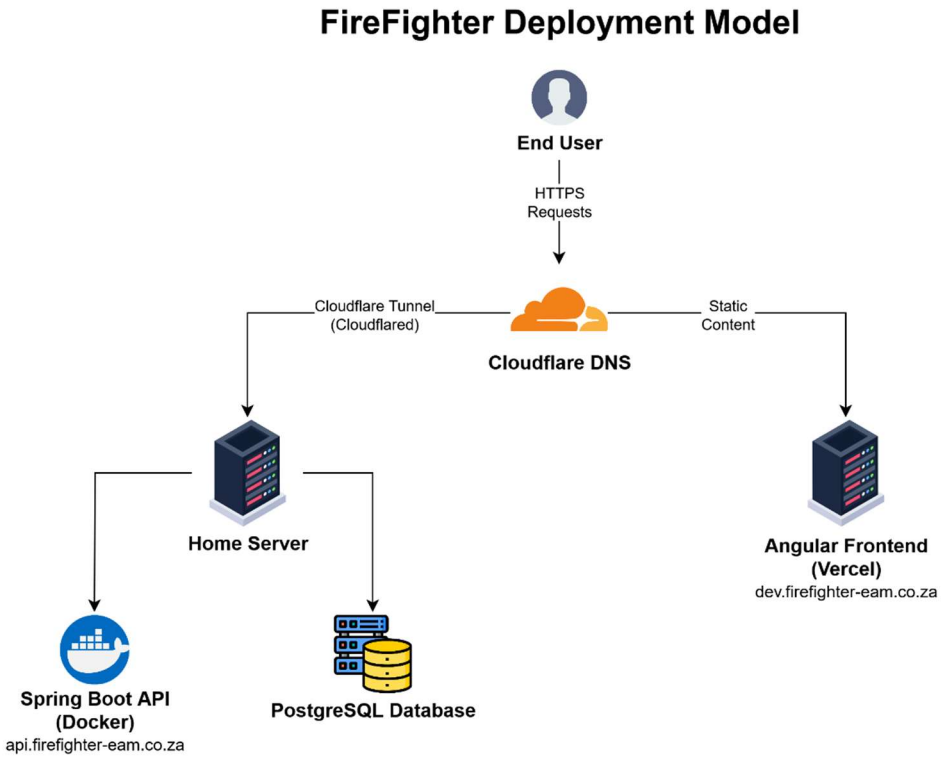
- **Public Cloud**: Frontend hosting (Vercel) + DNS/security (Cloudflare)
- **On-Premises**: Backend services (Home server)

Justification:

- **Cost Optimization**: Eliminates cloud compute costs for the backend and database.
- **Performance**: Provides low-latency, high-bandwidth communication between the backend API and database on the same local network.
- **Security & Control**: Sensitive data remains on-premises, and the attack surface is minimized by using a secure tunnel instead of open ports.
- **Global Reach**: Leverages enterprise-grade CDN and edge networking for the frontend to serve users worldwide with low latency.

9.2. Deployment Topology

Three-Tier Architecture with Secure Tunnelling:



9.2.1 Node Distribution

Component	Location	Specs
Angular Frontend	Vercel Global CDN	12 Edge locations
Spring Boot API	Docker on Home Server	4vCPU, 8GB RAM
PostgreSQL DB	Bare-metal (Same host)	256GB SSD

9.3. Deployment Tools & Platforms

9.3.1 Core Technology Stack

Technology	Version	Purpose	Deployment Role
Vercel	Platform	Frontend Hosting & CI/CD	Global CDN distribution of static Angular assets.
Docker	20.10+	Application Containerization	Isolates the Spring Boot API for consistent runtime environments and easy deployment.
Docker Compose	v2	Container Orchestration	Defines and manages the multi-container application service (API + DB planned).
Cloudflare Tunnel	(cloudflared)	Secure Reverse Proxy	Creates an outbound, encrypted tunnel from the on-prem server to Cloudflare's edge, exposing the API without public IPs or open firewall ports.
PostgreSQL	16.x	Relational Database	Persistent data storage running directly on the host OS for maximum I/O performance.
Ubuntu Server	22.04 LTS	Host Operating System	Provides a stable, secure base for the Docker engine and database.

9.4. Quality Requirements Support

9.4.1 Scalability

- **Frontend**: Automatic horizontal scaling via Vercel's global CDN network. Static assets are cached at edge locations closest to users.
- **Backend**: The containerized API is designed for horizontal scaling. The current deployment can be easily migrated to a Kubernetes cluster (k3s) by applying pre-written manifests to add replicas.
- **Database**: Utilizes HikariCP for efficient connection pooling. The architecture supports adding read replicas to distribute query load.

9.4.2 Reliability

Measure	Implementation	SLA
Redundant DNS	Cloudflare + secondary provider	99.99% uptime
Automated Backups	WAL-E + S3 integration	RPO < 1 hour
Health Monitoring	Prometheus + Grafana dashboard	24/7 alerts

Implementation Details:

- The Cloudflare Tunnel agent (cloudflared) is managed as a systemd service with automatic restart on failure.
- Database transactions are secured with Write-Ahead Logging (WAL), enabling point-in-time recovery.
- A Prometheus instance scrapes API health metrics, and alerts are configured in Grafana to notify administrators of downtime.

9.5. Failure Modes & Mitigation

Failure Scenario	Detection	Auto-Recovery Mechanism
Tunnel Disconnection	systemd watchdog timer	RestartSec=5 in service unit
Database Overload	Prometheus connection metrics	Auto-scale read replicas (pre-configured)
Frontend CDN Outage	Cloudflare Status API	Failover to static S3 bucket

Disaster Recovery Plan:

1. **Server Hardware Failure**: Restore the latest database backup to a new machine, update the Cloudflare Tunnel configuration to point to the new machine's IP, and redeploy the Docker container.
2. **Data Corruption**: Stop the application, restore the database from the latest nightly backup, and replay WAL logs to the point of failure.
3. **Cloudflare Outage**: Temporarily reconfigure DNS A records to point directly to the home server's public IP (requires manual port forwarding as a last resort).

Demo 2 (SRS Version 2)

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.

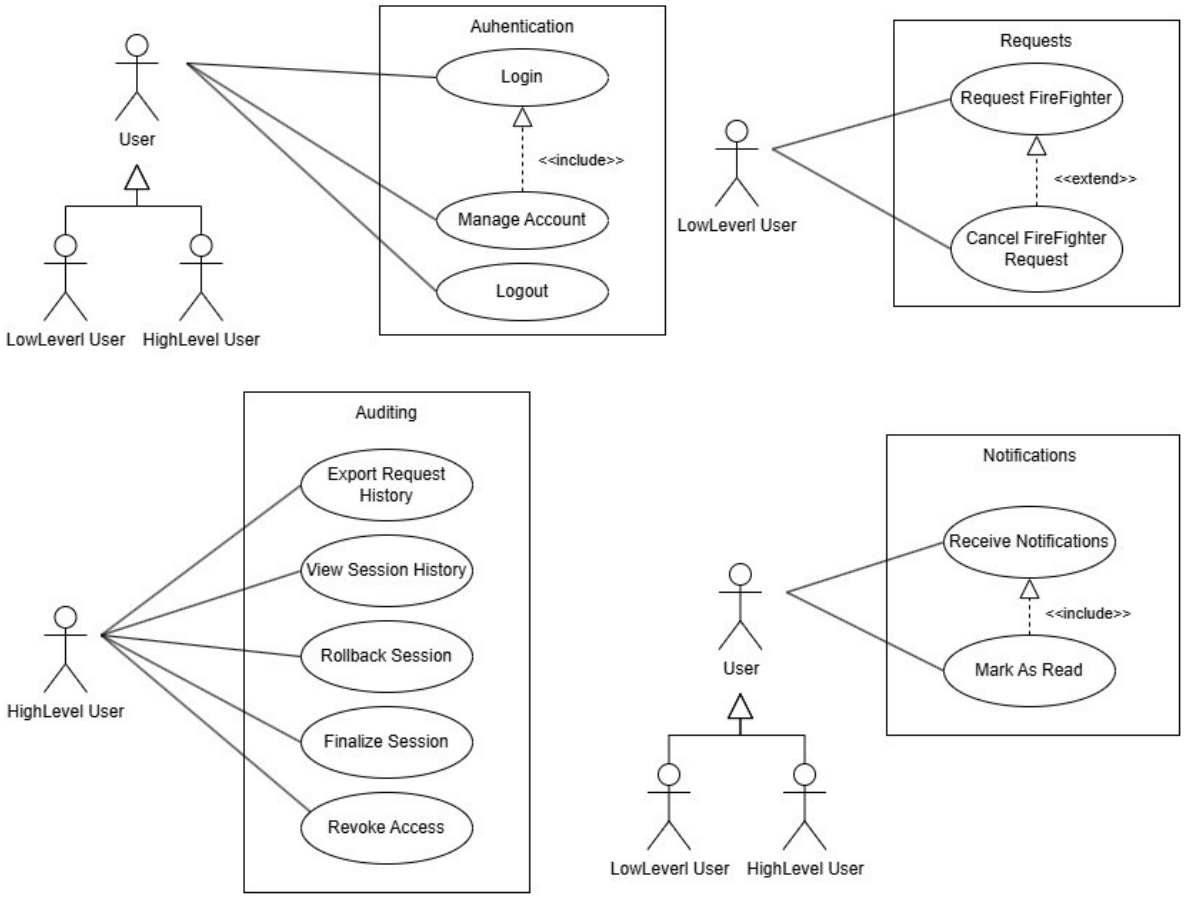
2.2.13 Help & Support

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
Auth Provider	Firebase OAuth2
Token Format	JWT (Firebase ID Token)
Header	Authorization: Bearer <ID_TOKEN>
Expiry	Set by Firebase (usually 1 hour)
Verification	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

Code	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

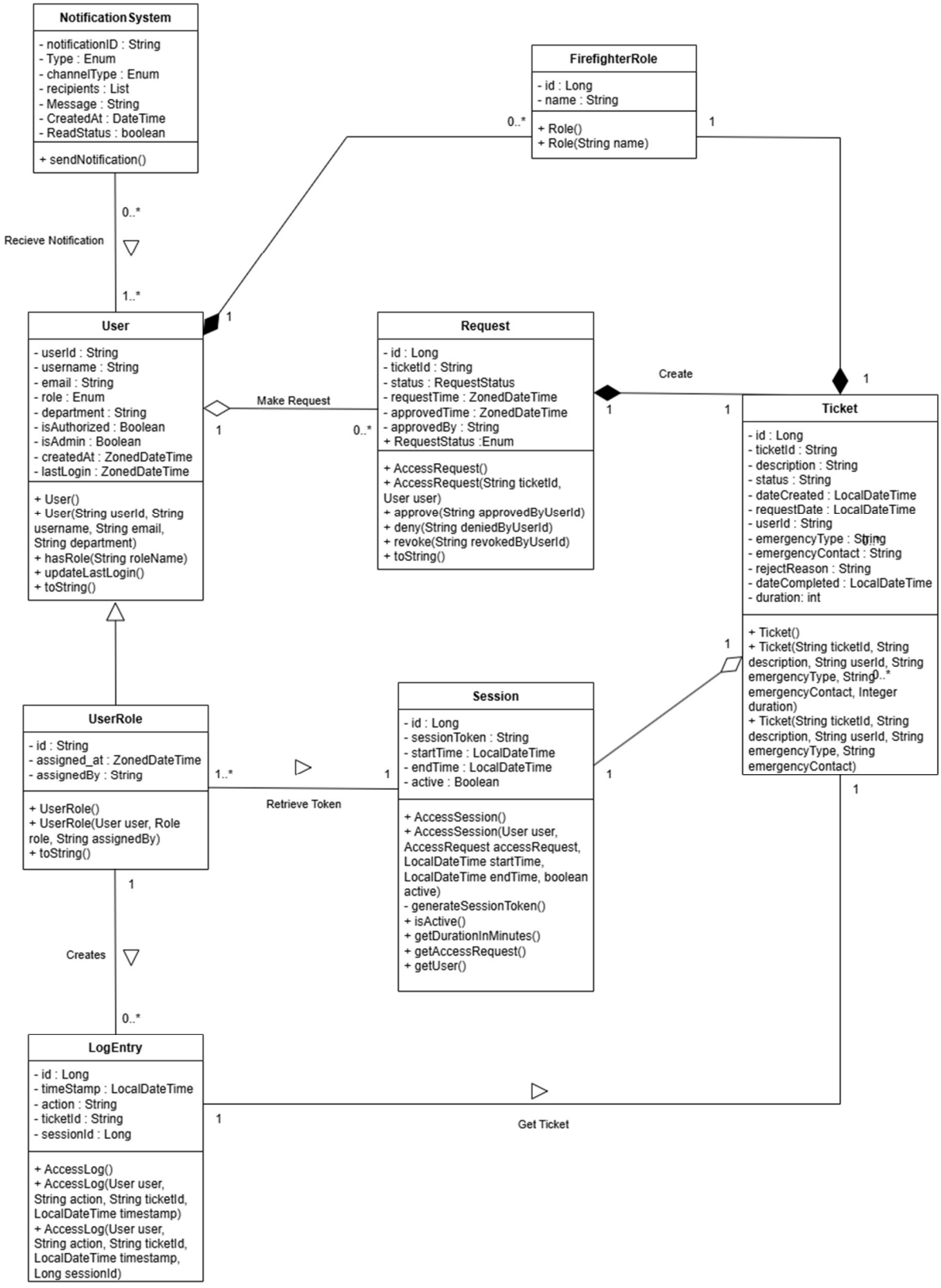
Endpoint: GET /api/tickets/ticket-id/{ticketId}
Response:

```
{
  "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
Frontend (Ionic)	RESTful API	HTTPS	Calls Spring Boot API via NGINX proxy
Backend (Spring)	REST Controller	HTTPS + JWT	Verifies Firebase tokens
DB (PostgreSQL)	JDBC / ORM	Localhost	Hibernate ORM
Logs	DB Table	-	Stored in access_logs table.
CI/CD	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.

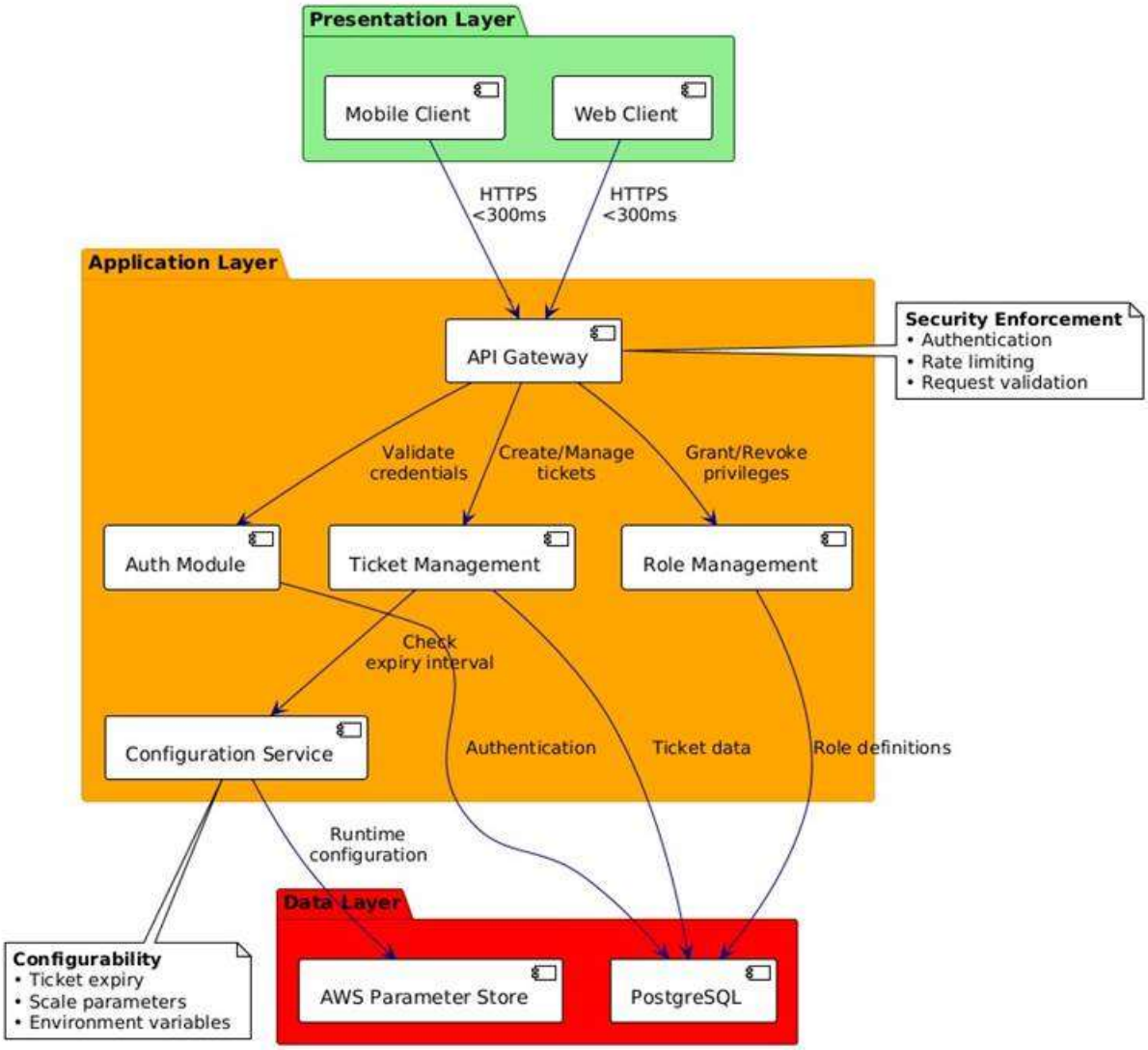
4. **Responsiveness**

   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
React Native	· Strongly encourages code reuse in both web & mobile applications · Large ecosystem	· Requires much configuration for web-mobile parity · Frequent plugin conflicts
Flutter	· Excellent performance and... · Conducive to an all-around good mobile experience	· Learning curve. Dev team has never used Dart · Unexemplary web support
Ionic + Angular + Capacitor	· Shared codebase between web & mobile · Rich ecosystem · Capacitor enables native device features	· Large bundle size (multiple installations and employed frameworks) · Learning curve for uninitiated team members

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
Node.js	· Lightweight · Easy to get started	· Not feature rich. · Requires additional setup for security and module patterns
Django	Exceptionally feature rich	· Incredibly slow Python runtime environment · Modularity comparable to (if not worse than) Spring Boot
Spring Boot	· Built in security · Excellent REST API support · Encourages modularity	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
PostgreSQL	Open source Wide AWS support Powerful	More complex setup compared to noSQL DBs
MySQL	Mature Wide support	Sparse features
MongoDB	Flexible 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
Maven	· Stable · Widely used in Java development · Frequently used in conjunction with Spring Boot	Verbose (easy human interaction)
Gradle	· Flexible · Fast project building	· Complicated configuration · Learning Curve
Ant	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:** ≥ 16.x / npm ≥ 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:** ≤ 2 s on a 4G connection
- **UI Responsiveness:** Interactive actions (theme switch, form validation) ≤ 200 ms
- **Mock Data Fetch:** ≤ 100 ms round-trip for local API stubs
- **Capacity Target:** Architected for ≥ 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:** ≥ 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