A close-up of a logo

Description automatically generated

**School of Computing and Digital Technologies**

**Software Architecture and Design**

**(55-608809)**

**Software Architecture and Design Document**

**Project:**  Complaint Management System (CMS)

|  |  |
| --- | --- |
| **Name** | **ID** |
| **Thomas Grout** | c3034731 |

**Date: 13/10/2025**

Table of Contents

[Introduction 3](#_Toc208957300)

[Overview 3](#_Toc208957301)

[Solution Architecture 4](#_Toc208957302)

[Non-Functional Requirements 4](#_Toc208957303)

[C4 Context Diagram (level 1) 4](#_Toc208957304)

[C4 Container Diagram (level 2) 4](#_Toc208957305)

[Technology Stack 4](#_Toc208957306)

[Solution Design 4](#_Toc208957307)

[User Interface Design 4](#_Toc208957308)

[C4 Component Diagram (level 3) 4](#_Toc208957309)

[C4 Code Diagrams (level 4) 4](#_Toc208957310)

[API endpoints 4](#_Toc208957311)

[Data Design 4](#_Toc208957312)

[Security Considerations 4](#_Toc208957313)

[Appendix 1: Transparency Declaration Statements 5](#_Toc208957314)

[Appendix 2: Refined User Stories 5](#_Toc208957315)

[Appendix 3: Use Cases 5](#_Toc208957316)

[Appendix 4: Architecture Decision Records 5](#_Toc208957317)

[Appendix 5: Incorporation of Formative Feedback 6](#_Toc208957318)

[Appendix 6: 7](#_Toc208957319)

[References 8](#_Toc208957320)

# Introduction

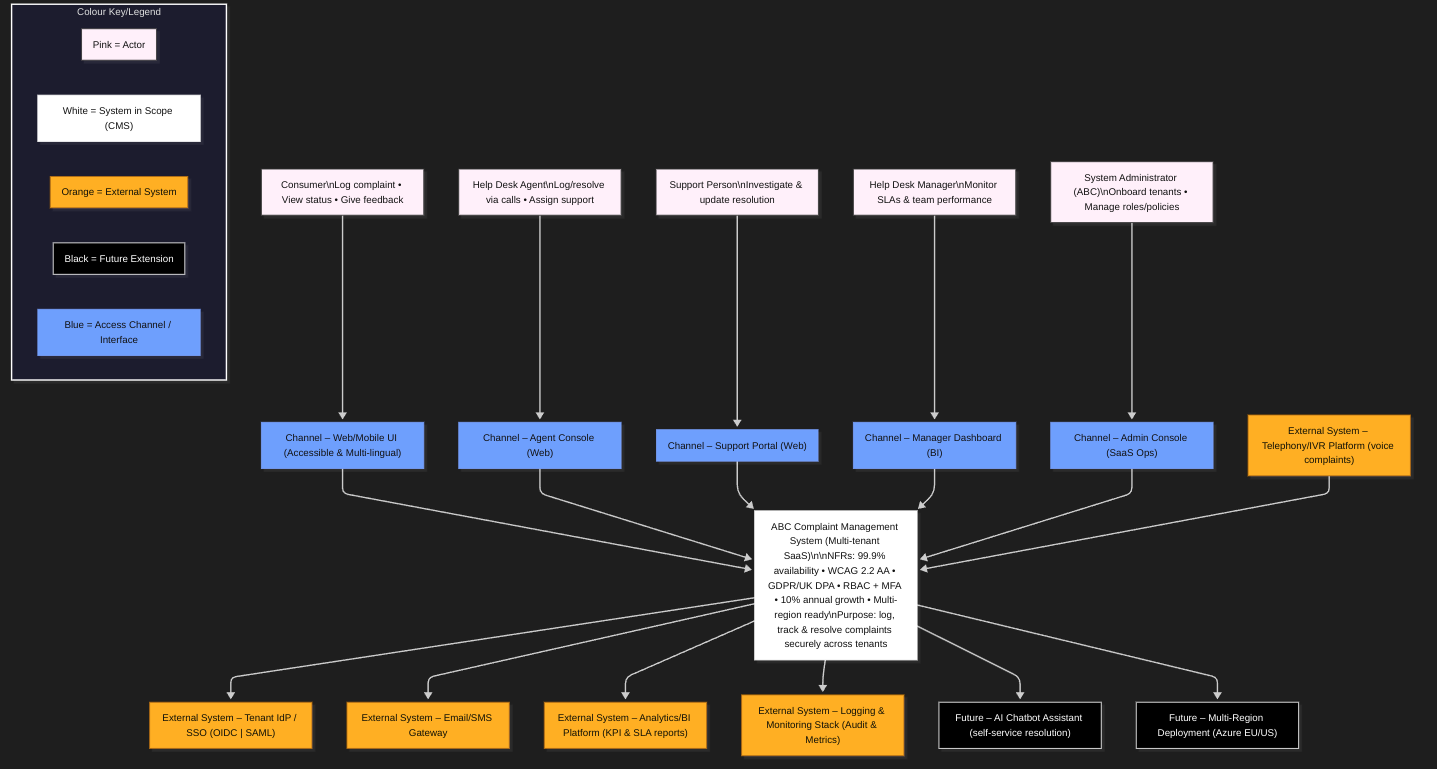
## Overview

# Solution Architecture

## Non-Functional Requirements

|  |  |  |  |
| --- | --- | --- | --- |
| **ID** | **Theme** | **Description** | **Priority (MoSCoW)** |
| NFR001 | Availability | 99.9% service availability; graceful degradation for notifications/analytics | Should |
| NFR002 | Scalability | Handle 20M+ users (bank-scale) with 10% YoY growth | Should |
| NFR003 | Performance | p95 < 300ms for read; < 700ms for create/update | Should |
| NFR004 | Security | OIDC/SAML, MFA, RBAC | Must |
| NFR005 | Privacy & Compliance | GDPR, UK DPA | Should |
| NFR006 | Accessibility | WCAG 2.2 AA | Must |
| NFR007 | Usability | Consistent UX everywhere with error clarity | Could |
| NFR008 | Multi-tenancy Isolation | No cross-tenant access - prove isolation | Must |
| NFR009 | Reliability | Silver (tier 2)[1] RTO = 4hrs, RPO = 1hr | Could |
| NFR010 | Observability | Traceability, diagnosability | Should |
| NFR011 | Extensibility | Future Chatbot + partner APIs | Would |
| NFR012 | Cost Efficiency | Achieve functionality with fewer resources, such as CPU cycles, memory, and storage. [2] | Would |

## C4 Context Diagram (level 1)

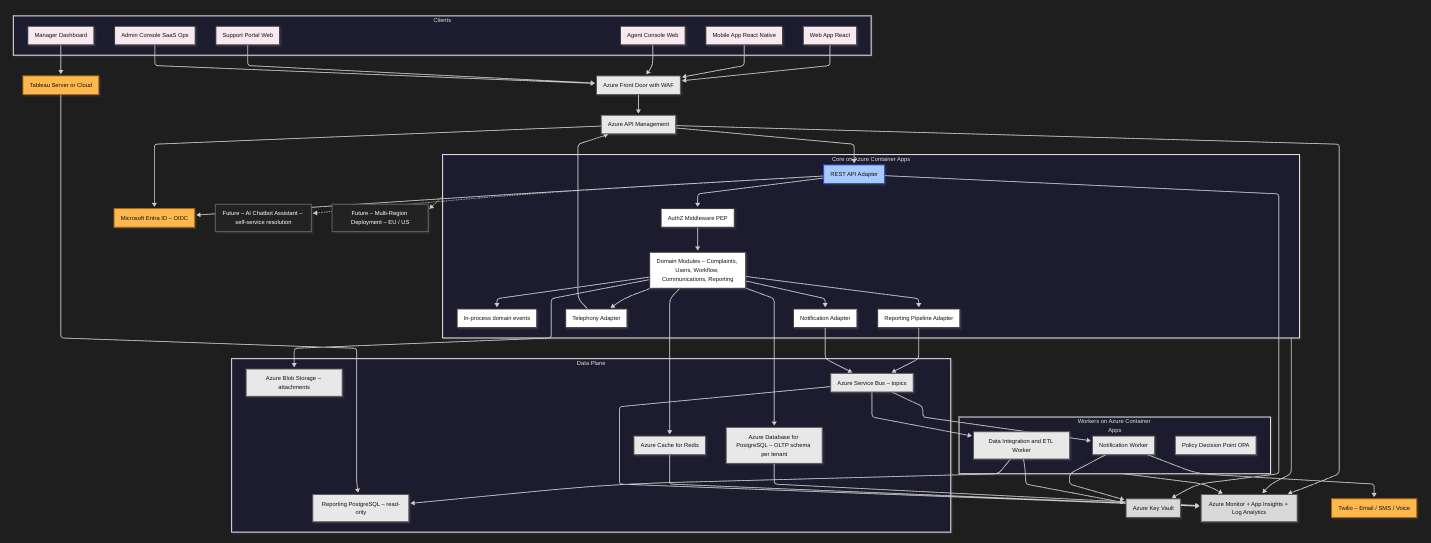


The Level-1 context diagram defines the operational boundary of the **ABC Complaint Management System (CMS).** The CMS operates as a multi-tenant Software-as-a-Service platform designed to mediate the complete lifecycle of customer complaints across diverse organisations, primarily banking and telecom companies. The system ensures security, scalability, performance and availability are met first and foremost, with an additional focus on accessibility, usability and privacy. Distinct user roles, i.e. consumer, help-desk agent, support engineer, manager, and system administrator interact with the system through dedicated access channels that reflect their functional responsibilities.

Each access channel, Web / Mobile UI, Agent Console, Support Portal, Manager Dashboard, and Admin Console, communicates with the CMS core via secure interfaces. External integrations with **OpenID Connect** and **SAML 2.0** identity providers enable authentication and multi-factor authentication, while connectivity to **telephone and IVR services**, **email / SMS gateways**, and **analytics and monitoring platforms** support performance management.

Planned extensions, including an **AI-driven self-service assistant** and **multi-region deployment**, are represented as future capabilities, maintaining architectural coherence with the system’s scalability and resilience objectives. In alignment with the **C4 Model**, this context view articulates the CMS as a single, cohesive system that orchestrates complaint management, communication, and reporting across securely isolated tenants, providing a clear foundation for the container and component levels.

## C4 Container Diagram (level 2)



This container diagram represents the CMS core as a modular monolith deployed on Azure Container Apps, structured to preserve a strict separation of concerns. The API Gateway mediates all inbound communication, enforcing tenant context, authentication, and authorisation through OpenID Connect before requests reach the core application. The system follows a hexagonal architecture, exposing well-defined ports for REST interfaces, event publishing, and external adapters. Within the data plane, persistence is isolated through PostgreSQL schemas per tenant with row-level security, supported by Redis for caching and Azure Blob Storage for unstructured data. Background workers manage asynchronous workloads through Azure Service Bus topics, decoupling side effects from user-facing transactions. Integrations with Twilio for notifications, Tableau for analytics, and Azure Key Vault for secret management are placed beyond the system’s trust boundary to uphold least-privilege principles. This configuration achieves operational clarity and scalability without incurring the coordination costs of microservices, providing an evolvable foundation consistent with contemporary architectural practice and the analytical standards required for this design study.

## Technology Stack

Front end

| Category | Technology | Purpose |
| --- | --- | --- |
| Web Interfaces | **React (TypeScript)** | Accessible single-page applications (WCAG 2.2 AA) |
| Mobile Application | **React Native** | Multilingual consumer mobile interface |
| Admin Interface | **React Admin** | Tenant onboarding and role management console |

Application

| Category | Technology | Purpose |
| --- | --- | --- |
| Language / Runtime | **Node.js** | Core runtime for the CMS |
| Framework | **Express** | RESTful API framework implementing the Hexagonal pattern |
| ORM | **TypeORM** | Database access with schema-per-tenant and row-level security |
| Event Handling | **BullMQ** | Redis-backed message queue for domain events and async tasks |
| Validation & API Spec | **Zod + OpenAPI (Swagger)** | Request validation and documentation |
| Testing | **Jest** | Unit and integration testing framework |

Auth

| Category | Technology | Purpose |
| --- | --- | --- |
| Authentication / Authorisation | **Microsoft Entra ID (OIDC / SAML 2.0)** | SSO, MFA, and token validation |
| Tenant Context | **TenantResolver Middleware** | Verifies tenant identity |
| Audit & Logging | **Azure Monitor** | Structured logs and audit trail |
| Secrets Management | **Azure Key Vault** | Secure storage |

Data

| Category | Technology | Purpose |
| --- | --- | --- |
| Primary Database | **Azure Database for PostgreSQL Flexible Server** | Multi-tenant OLTP with schema isolation and backups |
| Cache | **Azure Cache for Redis** | In-memory cache for sessions and CQRS projections |
| File Storage | **Azure Blob Storage** | Persistent object storage for attachments |
| Messaging / Events | **Azure Service Bus** | Event bus for asynchronous inter-module communication |
| Reporting Store | **Azure PostgreSQL (Read Replica)** | Read-optimised database for analytics |

Services

| Category | Technology | Purpose |
| --- | --- | --- |
| Notifications | **Twilio** | Email, SMS, and voice notifications |
| Analytics / BI | **Tableau** | Reporting and performance dashboards |
| Monitoring | **Azure Monitor** | Central telemetry and service health tracking |
| Identity Providers | **Microsoft Entra ID** | Tenant-level authentication and authorisation |

Deployment

| Category | Technology | Purpose |
| --- | --- | --- |
| Ingress & Gateway | **Azure Front Door + API Management** | Secure entry point, WAF, and token validation |
| Application Hosting | **Azure Container Apps** | Scalable, managed runtime for the CMS Core |
| Event / Queue Broker | **Azure Service Bus** | Reliable event distribution for background jobs |
| Observability | **Application Insights** | Distributed tracing, metrics, and diagnostics |
| Disaster Recovery | **Azure Backup + Geo-Replication** | High availability and fault tolerance |

CI/CD

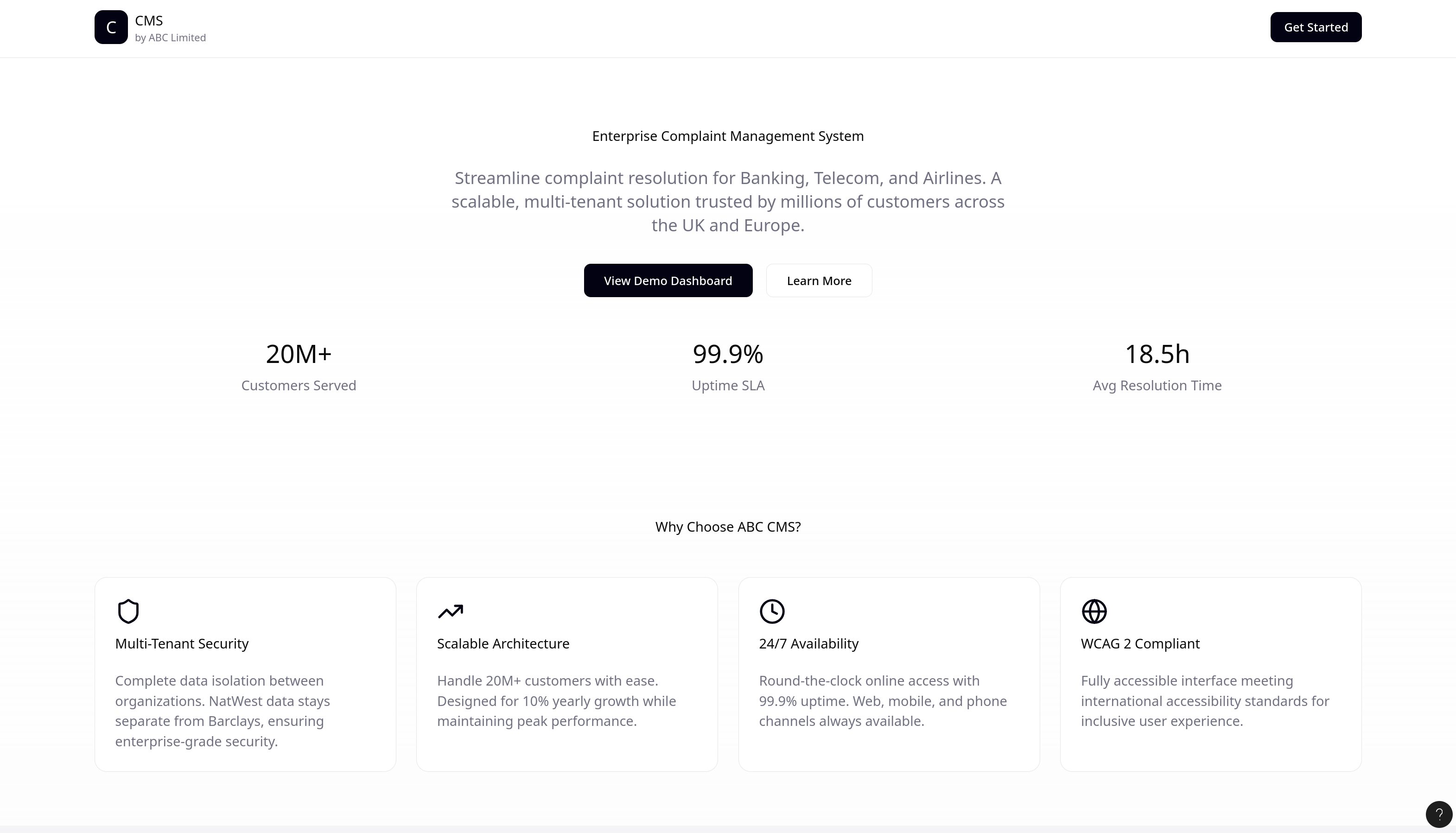
| Category | Technology | Purpose |
| --- | --- | --- |
| Version Control | **Git** | Source management and ADR repository |
| CI/CD Pipeline | **GitHub Actions** | Automated build, test, and deploy workflows |
| Containers | **Docker** | Packaging of CMS Core and worker services |
| Infrastructure as Code | **Terraform** | Reproducible provisioning of Azure resources |

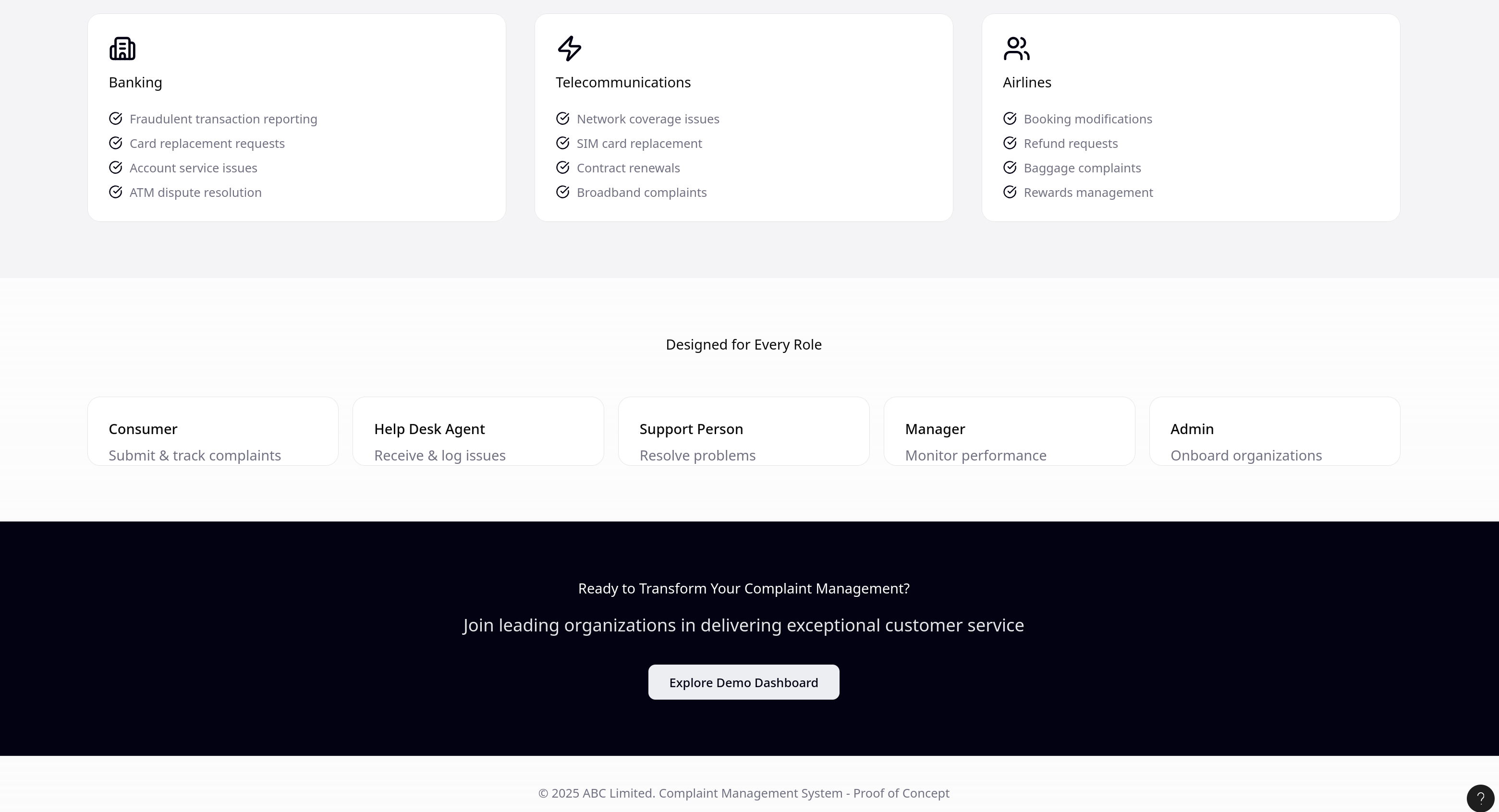
# Solution Design

## User Interface Design

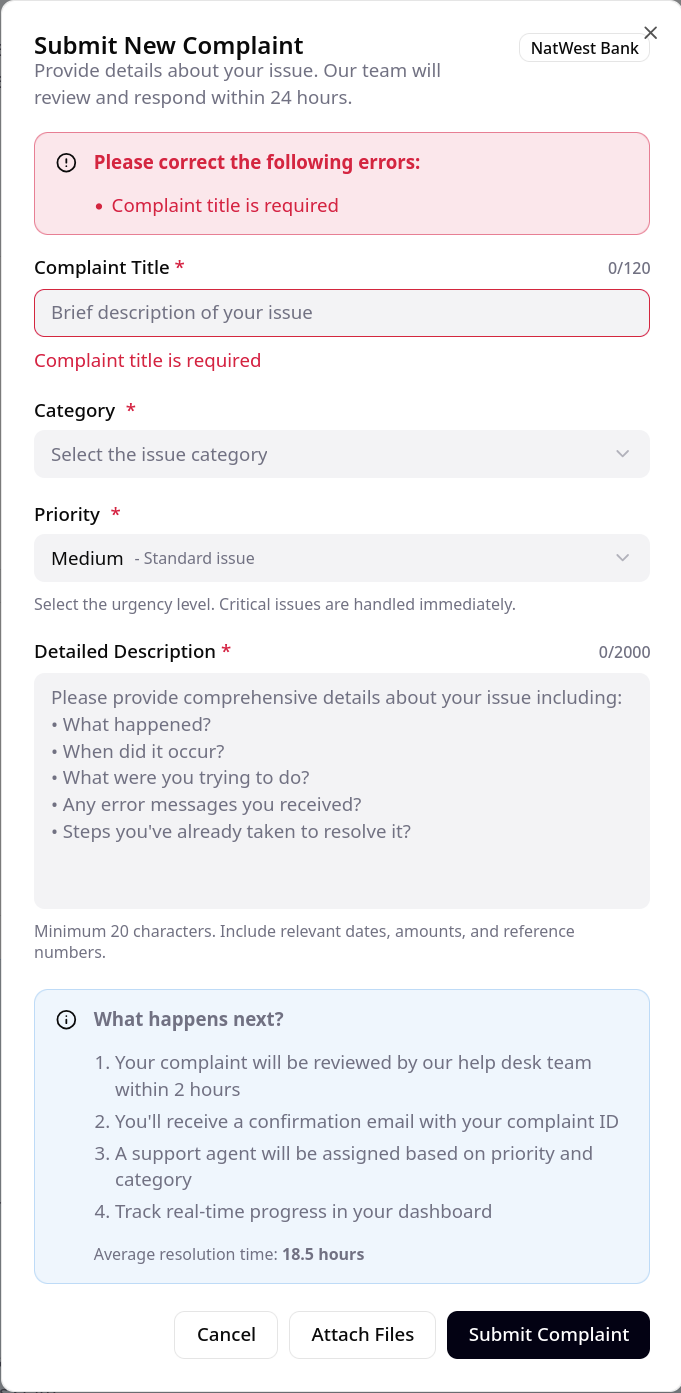
In this section you should include Wireframes. They should be consistent with the user stories and the non-functional requirements (especially usability requirements)

Home Page:



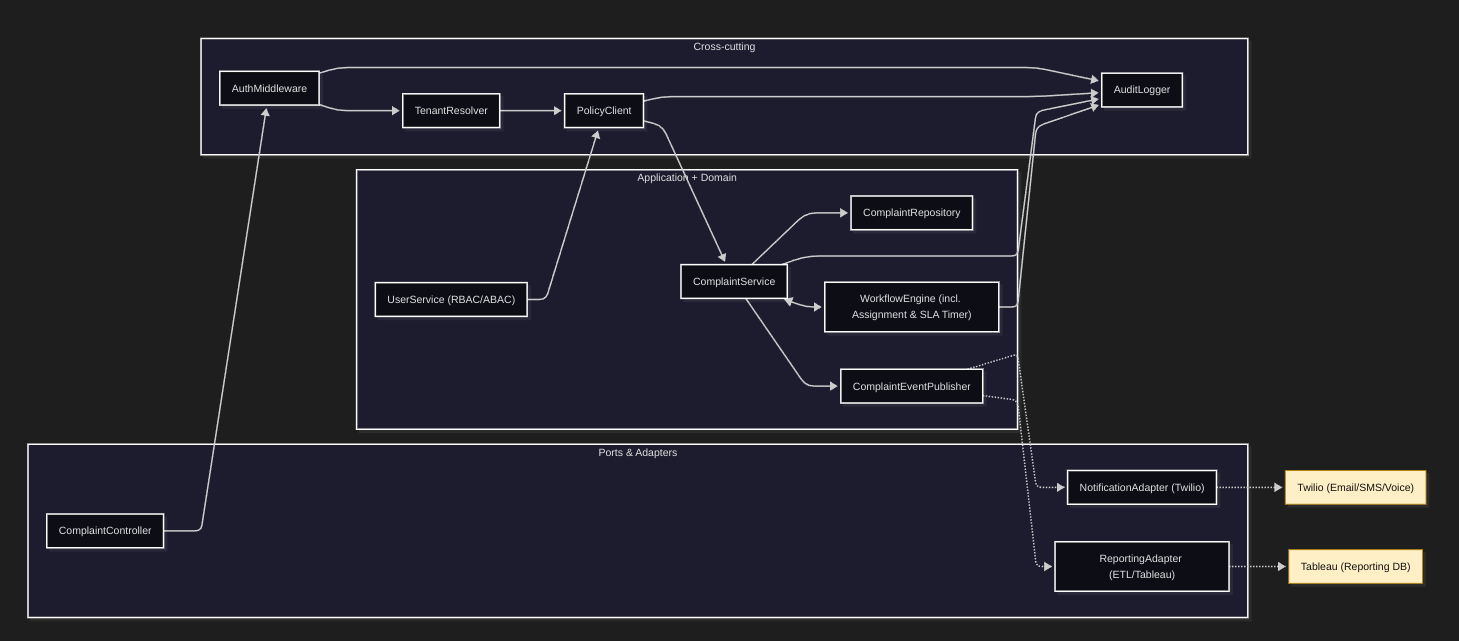


Complaint Form:



## 

## C4 Component Diagram (level 3)



This component diagram depicts the internal composition of the CMS Core container, structured as a modular monolith implemented according to the Hexagonal, or Ports and Adapters, architectural style. The adapter layer manages inbound REST interactions through the ComplaintController, routing validated requests to application services that encapsulate domain logic. Within the core, the ComplaintService coordinates persistence through the ComplaintRepository and orchestrates state transitions via the WorkflowEngine, which includes assignment handling and SLA timers. Domain events are published asynchronously by the ComplaintEventPublisher using BullMQ, enabling non-blocking communication with peripheral services. Cross-cutting components—AuthMiddleware, TenantResolver, PolicyClient, and AuditLogger—enforce identity, authorisation, multi-tenant isolation, and immutable audit logging. Outbound adapters, represented by the NotificationAdapter and ReportingAdapter, integrate securely with Twilio and Tableau while maintaining clear separation from the trusted runtime. The design maintains a single deployable unit with well-defined boundaries, achieving strong cohesion, isolation of concerns, and consistent enforcement of security and audit requirements. It aligns with the assessment criteria for a Level-3 view by presenting a coherent, pattern-compliant architecture that addresses both the functional workflow and the non-functional objectives of scalability, reliability, and compliance.

## C4 Code Diagrams (level 4)

* Structural diagram
* Behavioural diagram

## API endpoints

Document your APIs here if your design is service oriented. If your design is monolithic ignore this.

## Data Design

Include diagrams and description of the project data design.

## Security Considerations

Discuss security measures considered in your design; this may include authentication (SSO, multi factor authentication etc.), authorisation (role-based access) , encryption ( data level) etc.

# Appendix 1: Transparency Declaration Statements

If you take help of generative AI, please declare it here.

# Appendix 2: Refined User Stories

If you feel appropriate you may want to refine the user stories provided in the case study. You may take help of generative AI. If you take help, please acknowledge that. You can also refer to SHUDev Process template to refine user stories

# Appendix 3: Use Cases

Identify use cases from the information provided in the case study. You may take help of generative AI. If you take help, please acknowledge that.

# Appendix 4: Architecture Decision Records

Include one key ADR example here.

Add Link to GitHub repository that includes all project ADRs.

## 

# Appendix 5: Incorporation of Formative Feedback

Write here about evaluation of Formative Feedback and Actions Taken in Response to Feedback.

Table 1 - Weekly Feedback

|  |  |  |
| --- | --- | --- |
| **Week #** | **Feedback** | **Response** |
| **1** |  |  |
| **2** |  |  |
| **3** |  |  |
| **4** |  |  |
| **5** |  |  |
| **6** |  |  |
| **7** |  |  |
| **8** |  |  |
| **9** |  |  |
| **10** |  |  |
| **11** |  |  |
| **12** |  |  |

# Appendix 6:

Use if needed. You can include github link , information needed to augment your architecture or design or anything else you may feel relevant.

# References

**1.** RPO and RTO: What Are They and How to Calculate Them

UniTrends

<https://www.unitrends.com/blog/rpo-rto/>

2021

2. Architecture strategies for optimizing code costs

Microsoft

<https://learn.microsoft.com/en-us/azure/well-architected/cost-optimization/optimize-code-costs>

2023