

Faculty of Engineering – Ain Shams University

Computer and Systems Engineering Department

zGate Gateway: A Zero Trust Database Access Proxy

Graduation Project Thesis

Supervisor:

Dr. Mohammed Sobh

Academic Year 2025–2026

Acknowledgments

We would like to thank...

Abstract

This project introduces a Zero Trust–based database access gateway (SecureDB Gateway)...

Contents

1 Team Information	12
1.1 Team Members	12
1.2 Roles & Responsibilities	12
2 Introduction & Problem Definition	13
2.1 Introduction	13
2.2 Problem Statement	14
2.2.1 The Current State of Database Access Management	14
2.2.2 Critical Security Vulnerabilities	14
2.2.3 Impact and Consequences	15
2.2.4 The Need for Zero Trust Database Access	15
2.3 Gaps In Existing Solutions	16
2.3.1 Perimeter-Based Security	16
2.3.2 VPN-Based Access	16
2.3.3 Bastion Hosts and Jump Servers	17
2.3.4 Database Native Access Controls	17
2.3.5 Privileged Access Management (PAM) Solutions	18
2.3.6 Database Activity Monitoring (DAM)	18
2.3.7 The Fundamental Gap	19
2.4 Why Zero Trust for Databases is Different	19
2.4.1 Core Zero Trust Principles Applied to Databases	19
2.4.2 The Zero Trust Database Gateway Architecture	19
2.4.3 Query-Level Data Protection	20
2.4.4 Complete Auditability and Traceability	21
2.4.5 What zGate Implements	21
2.4.6 Operational Advantages	22
2.4.7 Addressing the Gaps	22
3 Project Definition	24
3.1 Project Definition and Scope	24
3.1.1 System Components	24
3.1.2 Scope Boundaries	26

3.2	Objectives	27
3.2.1	Core Security Objectives	27
3.2.2	Operational Objectives	28
3.2.3	Academic and Technical Learning Objectives	29
4	Requirements Engineering	31
4.1	Functional Requirements	31
4.1.1	FR-1: User Authentication & Authorization	31
4.1.2	FR-2: Database Connection Management	31
4.1.3	FR-3: Policy Engine	31
4.1.4	FR-4: Command Line Interface	31
4.1.5	FR-5: Web User Interface	32
4.1.6	FR-6: Protocol Handling	32
4.1.7	FR-7: Audit & Logging	32
4.2	Non-Functional Requirements	32
4.2.1	NFR-1: Security	32
4.2.2	NFR-2: Performance	33
4.2.3	NFR-3: Scalability Targets	33
4.2.4	NFR-4: Reliability	33
4.2.5	NFR-5: Usability	33
4.2.6	NFR-6: Maintainability	33
4.2.7	NFR-7: Portability	34
4.2.8	NFR-8: Compatibility	34
4.3	Actors & Use Cases	34
4.3.1	System Actors	34
4.3.2	Use Case Catalog	37
4.4	Use Case Diagrams	45
4.4.1	End User Case Diagrams	45
4.4.2	Administrator Use Case Diagrams	45
4.4.3	Complete System Use Case Diagrams	45
4.4.4	System Components Interaction	46
4.5	User Stories	46
4.5.1	End User Stories	46
4.5.2	Administrator Stories	49
4.5.3	Super Administrator Stories	55
4.5.4	System Stories (Non-Interactive)	56
5	Proposed Solution	59
5.1	Solution Overview	59

5.1.1	The Core Problem	59
5.1.2	The zGate Solution	59
5.2	Core Components	60
5.2.1	zGate Server (Backend)	60
5.2.2	zGate CLI (Client)	61
5.2.3	zGate WebUI (Admin Dashboard)	63
5.3	Key Technologies & Design Decisions	65
5.4	Security Architecture	65
5.4.1	Security Layers	66
5.4.2	Security Guarantees	67
5.5	Advantages of the Proposed Solution	67
6	Alignment with International Standards	69
6.1	PCI DSS	69
6.2	HIPAA	69
6.3	GDPR	70
6.4	ISO 27001	70
7	Competitor & Market Analysis	71
7.1	Competitor Analysis	71
7.1.1	Comparison Table	71
7.1.2	What Competitors Lack	71
7.2	Market Research	71
7.2.1	Market Overview	71
7.2.2	Zero Trust Demand	71
7.2.3	Market Challenges & Needs	71
7.2.4	Regulatory Drivers	71
7.2.5	Trends & Opportunities	71
7.2.6	Landscape Summary	71
8	Scientific Research & Literature Review	72
8.1	Generative AI-Enhanced Cybersecurity Framework for Enterprise Data Privacy Management	72
8.1.1	Purpose of the Study	72
8.1.2	Framework Overview	72
8.1.3	Implementation & Experiments	73
8.1.4	Privacy & Security Features	73
8.1.5	Contributions to the Paper	74
8.1.6	Advantages & Limitations	74
8.1.7	Relevance to Our Project	74

8.2	The Significance of Artificial Intelligence in Zero Trust Technologies: A Comprehensive Review	75
8.2.1	Problem Addressed	75
8.2.2	Methodology	75
8.2.3	Key Contributions of AI to Zero Trust	75
8.2.4	Findings	76
8.2.5	Comparison with Traditional Methods	76
8.2.6	Relevance to Our Project	76
8.3	Securing Digital Identity in the Zero Trust Architecture: A Blockchain Approach to Privacy-Focused Multi-Factor Authentication	77
8.3.1	Problem Addressed	77
8.3.2	Research Goals	77
8.3.3	Proposed System	77
8.3.4	Experimental Results	77
8.3.5	Key Findings and Contributions	78
8.3.6	Real-World Applications	78
8.3.7	Relevance to Our Project	78
8.3.8	Conclusion	78
8.4	Paper 4	79
8.5	Paper 5	79
8.6	Paper 6	79
8.7	Paper 7	79
8.8	Research References	79
9	Technical Background	80
10	System Architecture	81
10.1	High-Level Architecture Diagram	81
10.2	Main System Components	81
10.3	Component Communication Flow	81
10.4	Tech Stack Summary	81
11	Detailed Architecture of the Proxy	82
11.1	Connection Lifecycle	82
11.1.1	Lifecycle States	84
11.1.2	Detailed Lifecycle Flow	85
11.2	Authentication Flow	85
11.2.1	Authentication Architecture	86
11.2.2	Token Structure	86
11.2.3	Token Refresh Flow	87

11.3	Query Filtering Flow	87
11.3.1	Temporary User Creation	90
11.4	Policy Enforcement Flow	90
11.4.1	Policy Architecture	90
11.4.2	Real-Time Policy Evaluation	91
11.4.3	Policy Enforcement Points	91
11.4.4	End-to-End Flow Narrative	91
12	High-Level Data Flow Diagrams	94
12.1	Authentication Flow Diagram	94
12.2	Query Filtering Diagram	94
12.3	Logging & Auditing Flow Diagram	94
13	Technology Justification	95
13.1	Why Go	95
13.2	Why Node.js / TS / React	95
13.3	Why mTLS (and why TCP is temporary)	95
13.4	Why SQLite / Internal Storage	95
13.5	Design Decision Summary	95
14	Prototype – Semester 1	96
14.1	Implemented Features	96
14.2	Screenshots (CLI & Dashboard)	96
14.3	What Works vs What Doesn't	96
14.4	Technical Decisions Made	96
14.5	Implementation Challenges	96
15	Development Methodology	97
15.1	Agile Scrum Framework	97
15.2	Meeting Structure	97
15.2.1	Weekly Kick-off Meeting	97
15.2.2	Daily Stand-up Meeting	98
15.2.3	Sprint Review (Weekly Supervisor Meeting)	98
15.3	Collaboration Tools	98
15.4	Documentation & Observability	99
16	Task Tracking	100
16.1	Team Task Tracking (Actual Examples)	100
16.2	Supervisor Tracking Logs	100
16.3	Blockers, Risks & Resolution Notes	100

17 Milestones	101
17.1 Term 1 Milestone Roadmap	101
17.1.1 Milestone 1	101
17.1.2 Milestone 2	101
17.1.3 Milestone 3	101
17.1.4 Milestone 4	101
17.1.5 Milestone 5	101
17.2 Timeline Chart (Gantt-like)	101
18 Threat Model & Security Considerations	102
18.1 Threat Model	102
18.2 Risks & Attack Vectors	102
18.3 Mitigation Techniques	102
18.4 Why Zero Trust is Needed	102
19 Roadmap for Term 2	103
19.1 Remaining Features	103
19.2 Architecture Improvements	103
19.3 Performance Goals	103
19.4 Testing & Validation Plan	103
20 Team Contribution	104
20.1 Overview of Contribution Approach	104
20.2 Individual Contributions	104
21 Expected Outcomes	105
22 Conclusion	106
22.1 Restated Purpose	106
22.2 Summary of Achievements	106
22.3 Importance & Contribution	106
22.4 Transition to Next Semester	106
A Glossary	109
B Dashboard Mockups	110

List of Figures

2.1 what zGate offers	22
4.1 Overview of system actors: End Users, System Administrators, Super Administrators, and Backend Databases	34
4.2 End user workflows: authentication, database listing, connection, session management, and logout	45
4.3 Administrator functions: database configuration, user and role management, session monitoring, and audit review	45
4.4 Complete system use case diagram showing all actor interactions within the zGate Gateway	45
4.5 System component interactions showing data flow between CLI, Gateway Server, Web Dashboard, Policy Engine, and Backend Databases	46
5.1 zgate proposed solution architecture	60
5.2 zGate backend architecture showing authentication, policy engine, and proxy layers	60
5.3 zGate CLI workflow illustrating token management and database access process	62
5.4 zGate technology stack	65
5.5 Multi-layered security architecture in zGate platform	66
6.1 zGate alignment with PCI DSS requirements for access control, authentication, and audit logging	69
6.2 zGate alignment with GDPR requirements for data security, access control, audit records, and breach detection	70
11.1 system components	82
11.2 connection lifecycle states	84
11.3 Detailed connection lifecycle flow	85
11.4 Authentication architecture	86
11.5 Token refresh flow	87
11.6 Permission levels: read, write, admin	88
11.7 Permission enforcement flow	89
11.8 Temporary user creation flow	90

11.9 Policy enforcement architecture	90
11.10 Real-time policy evaluation	91
11.11 Policy enforcement points in the connection lifecycle	91

List of Tables

1.1	Team Members Information	12
2.1	Comparison of Traditional Limitations vs. Zero Trust Solutions	23
5.1	Security threats and their mitigations in zGate	67
11.1	MSSQL permission mapping	89
11.2	MySQL permission mapping	89

1. Team Information

1.1 Team Members

Name	ID	LinkedIn
Moustafa Ahmed	2100467	Moustafa Hashem
Kareem Ehab	2100913	Kareem Ehab
Hana Shamel	2100468	Hana Shamel
Karen Maurice	2100748	Karen Maurice
Michael George	2100709	Michael George
Mayar Walid	2100953	Mayar Walid
Rodina Mohammed	2100754	Rodina Mohammed

Table 1.1: Team Members Information

1.2 Roles & Responsibilities

2. Introduction & Problem Definition

2.1 Introduction

In the modern data-driven enterprise landscape, databases serve as the foundation for critical business operations, storing sensitive customer information, financial records, intellectual property, and operational data. However, the traditional approaches to database access management have not evolved at the same pace as the sophistication of cyber threats and the complexity of organizational structures. Development teams, database administrators, data analysts, and various other technical personnel routinely require direct database access to perform their duties, creating significant security challenges that existing solutions fail to adequately address.

zGate is a Zero Trust database access gateway designed to fundamentally transform how organizations manage, secure, and audit database access. Built on the principles of Zero Trust security architecture, zGate operates as an intelligent intermediary layer between users and database systems, enforcing identity-based access control, implementing dynamic query-level authorization, and ensuring complete auditability of all database operations. The system comprises three integrated components: a high-performance gateway server that intercepts and controls all database traffic, a command-line interface for streamlined user interactions, and a comprehensive web-based administration dashboard for policy management and monitoring.

By eliminating the need for developers and analysts to possess or handle production database credentials directly, zGate addresses the critical security gap that has led to numerous data breaches and compliance failures across industries. The system enforces the principle of least privilege through role-based access control policies, generates transient session-specific credentials, and provides real-time query filtering and data masking capabilities to protect sensitive information even when legitimate users are accessing the database.

2.2 Problem Statement

2.2.1 The Current State of Database Access Management

Contemporary organizations face a fundamental security dilemma in database access management. On one hand, operational efficiency demands that developers, data engineers, DevOps teams, and analysts have timely access to databases for development, troubleshooting, analytics, and maintenance activities. On the other hand, granting such access using traditional methods introduces severe security vulnerabilities that expose organizations to data breaches, insider threats, and regulatory non-compliance.

2.2.2 Critical Security Vulnerabilities

The prevailing practices in database access management present several critical vulnerabilities:

Static Credential Proliferation: Organizations typically rely on shared or long-lived static credentials that are distributed among team members. These credentials often appear in configuration files, environment variables, documentation, and even code repositories, creating numerous attack vectors. Once compromised, these credentials provide unrestricted access until manually rotated—a process that is infrequent and operationally disruptive.

Lack of Accountability and Auditability: When multiple users share the same database credentials, individual accountability becomes impossible. Security teams cannot determine which specific user executed a particular query, making post-incident investigation extremely difficult and enabling malicious insiders to operate with impunity. Traditional database audit logs capture the database username but not the actual human identity behind the action.

Excessive Privilege and Unrestricted Access: Developers and technical personnel are often granted broader database permissions than necessary for their specific tasks. A developer needing read-only access to a single table might receive full database access simply because granular permission management is too complex or time-consuming to implement properly. This violates the principle of least privilege and dramatically expands the attack surface.

Inadequate Protection of Sensitive Data: Even legitimate users with proper authorization may inadvertently expose sensitive information such as personally identifiable information (PII), financial data, or health records. Current systems lack the capability to dynamically mask or redact sensitive fields based on user identity and context, forcing organizations to choose between operational efficiency and data protection.

Insider Threat Vulnerability: Trusted insiders with legitimate database access

represent one of the most significant security risks. Whether through malicious intent, negligence, or social engineering, insiders can exfiltrate sensitive data, manipulate records, or cause operational disruptions with minimal detection risk under current access paradigms.

2.2.3 Impact and Consequences

These vulnerabilities have tangible consequences:

- **Data Breaches:** Compromised credentials or malicious insiders lead to unauthorized data exfiltration
- **Regulatory Non-Compliance:** Failure to meet requirements of GDPR, HIPAA, PCI-DSS, and other frameworks
- **Financial Losses:** Direct costs from breaches, regulatory fines, and operational disruptions
- **Reputational Damage:** Loss of customer trust and competitive disadvantage
- **Operational Inefficiency:** Cumbersome manual processes for credential management and access provisioning

2.2.4 The Need for Zero Trust Database Access

The transition to Zero Trust architecture in network and application security has demonstrated the effectiveness of “never trust, always verify” principles. However, database access has remained largely unchanged, still operating under implicit trust models. There is an urgent need for a solution that:

- Eliminates direct credential exposure by ensuring users never handle production database credentials
- Enforces identity-based access control tied to organizational identity management systems
- Implements dynamic, session-specific authorization rather than static permissions
- Provides query-level policy enforcement to control what operations each user can perform
- Ensures complete auditability with full traceability of every database operation to individual users

- Supports data-level security through dynamic masking and filtering of sensitive information
- Maintains operational efficiency without imposing excessive burden on legitimate users

2.3 Gaps In Existing Solutions

Organizations have historically relied on several security approaches to protect database access, each with significant limitations that fail to address the core vulnerabilities outlined previously.

2.3.1 Perimeter-Based Security

Traditional perimeter security operates on the assumption that threats exist outside the network boundary while everything inside is trustworthy. Firewalls, network segmentation, and IP whitelisting control which systems can reach database servers.

Limitations:

- **Lateral Movement:** Once an attacker breaches the perimeter (through phishing, compromised endpoints, or insider access), they can move freely within the network and access databases directly
- **No Identity Verification:** Perimeter controls verify network location, not user identity. Anyone on an authorized network or VPN can access databases
- **Coarse-Grained:** Controls apply at the network level, not at the query or data level. A user with network access has unrestricted database access
- **Cloud Incompatibility:** Modern cloud architectures and remote work arrangements render network perimeters increasingly porous and difficult to define

2.3.2 VPN-Based Access

Virtual Private Networks extend the corporate network to remote users, creating an encrypted tunnel that makes remote devices appear as if they're on the internal network.

Limitations:

- **All-or-Nothing Access:** VPN grants network-level access to all resources within its scope. A user connected via VPN can potentially access any database on that network segment

- **Shared Credentials Still Required:** VPN only solves the network connectivity problem; users still need database credentials, perpetuating the static credential problem
- **No Query-Level Control:** VPN cannot inspect, filter, or control database queries based on content or context
- **Session Persistence:** VPN sessions often remain active for extended periods, providing prolonged access windows for potential compromise
- **No Audit Trail:** VPN logs show connection events but provide no visibility into actual database operations performed

2.3.3 Bastion Hosts and Jump Servers

Organizations deploy intermediate servers that users must connect to before accessing databases, providing a centralized access point and audit logging.

Limitations:

- **Credential Exposure:** Users still retrieve and use actual database credentials, even if through a bastion host
- **Limited Policy Enforcement:** Bastion hosts log connections but typically cannot enforce query-level policies or filter sensitive data
- **Operational Overhead:** Requires maintaining additional infrastructure and managing access to the bastion itself
- **Session Recording Limitations:** While some bastion solutions record sessions, they provide after-the-fact forensics rather than real-time policy enforcement
- **Circumvention Risk:** Technical users can potentially bypass bastion hosts if they obtain credentials through other means

2.3.4 Database Native Access Controls

Modern databases include built-in authentication, authorization, and audit logging capabilities.

Limitations:

- **Complex Management:** Managing granular permissions across multiple databases and numerous users becomes administratively prohibitive at scale
- **Static Permissions:** Database roles and privileges are typically static and don't adapt to context (time, location, purpose)

- **Shared Account Pattern:** The complexity of per-user account management often leads organizations to share credentials anyway
- **Limited Masking Capabilities:** While some databases support row-level security and column masking, these features are database-specific, complex to configure, and inflexible
- **No Centralized Policy:** Each database system has its own permission model, preventing consistent policy enforcement across heterogeneous environments

2.3.5 Privileged Access Management (PAM) Solutions

PAM systems manage and audit privileged account credentials, often providing password vaulting, session recording, and credential rotation.

Limitations:

- **Still Credential-Based:** PAM distributes credentials to users, even if temporarily. Users still handle and potentially misuse actual database passwords
- **Session-Level, Not Query-Level:** PAM typically operates at the session level, recording entire sessions but not enforcing policies on individual queries
- **Limited Data Protection:** PAM cannot dynamically mask sensitive data fields based on user identity or query context
- **Operational Friction:** The check-out/check-in process for credentials adds significant overhead to developer workflows
- **PostgreSQL/MySQL Limitations:** Many PAM solutions were designed for privileged OS access and offer limited database protocol support

2.3.6 Database Activity Monitoring (DAM)

DAM solutions monitor and alert on database activity by analyzing network traffic or database logs.

Limitations:

- **Reactive, Not Preventive:** DAM detects suspicious activity after it occurs rather than preventing unauthorized actions proactively
- **No Access Control:** DAM cannot prevent users from executing queries; it only observes and reports
- **Alert Fatigue:** Organizations receive numerous alerts but lack the ability to block malicious activity in real-time

- **Identity Blindness:** DAM sees database usernames but often cannot tie actions to actual human identities when credentials are shared

2.3.7 The Fundamental Gap

All existing approaches share a common fundamental flaw: they separate authentication/authorization from the actual data access point.

Users authenticate to VPNs, bastion hosts, or PAM systems, but ultimately receive raw database credentials and connect directly to databases. This creates an uncontrolled gap where policy enforcement, audit logging, and data protection cannot be reliably applied.

Additionally, none of these solutions adequately address the credential exposure problem. Whether stored in password vaults, configuration files, or manually entered, database credentials exist outside the security boundary and can be extracted, shared, or misused by authorized users.

2.4 Why Zero Trust for Databases is Different

Zero Trust database access represents a paradigm shift that fundamentally reimagines how database security should operate. Rather than attempting to secure the perimeter or audit after the fact, Zero Trust embeds security directly into the data access path itself.

2.4.1 Core Zero Trust Principles Applied to Databases

Never Trust, Always Verify: Every database access request is authenticated and authorized in real-time, regardless of network location or previous access history. There is no concept of “trusted internal network.”

Least Privilege Access: Users receive the minimum necessary permissions for their specific task at a specific moment. Access rights are dynamically evaluated based on identity, role, context, and policy.

Assume Breach: The architecture assumes that credentials may be compromised and that internal threats exist. Therefore, every query is inspected and controlled, and no user ever possesses credentials that could be misused outside the controlled gateway.

2.4.2 The Zero Trust Database Gateway Architecture

Unlike traditional solutions that operate adjacent to database access, a Zero Trust gateway like zGate becomes the exclusive access point for all database operations. This architectural position enables capabilities impossible with peripheral solutions.

Identity-Based Authentication: Users authenticate using their organizational identity (JWT tokens, SSO integration) rather than database credentials. Authentication is tied to the specific human or service, eliminating shared accounts and enabling true accountability.

Credential Abstraction: The gateway maintains actual database credentials internally. Users never see, handle, or transmit production database passwords. Even if a user’s authentication token is compromised, the attacker gains no direct database access—they must still pass through the gateway’s policy enforcement.

Protocol-Aware Interception: By implementing native database protocols (MySQL, PostgreSQL, MSSQL, etc.), the gateway can parse and inspect every query at the SQL level. This enables surgical policy enforcement that perimeter tools cannot achieve:

- Block specific SQL commands (DROP, DELETE) based on user role
- Restrict queries to specific tables or schemas
- Prevent unauthorized joins or subqueries
- Control result set size and query execution time

Session-Specific Access: Each connection through the gateway represents a distinct, auditable session tied to a specific user identity. Sessions are short-lived, context-aware, and can be terminated immediately if suspicious activity is detected.

Dynamic Policy Enforcement: Policies are evaluated in real-time for every query based on:

- User identity and assigned roles
- Target database and table
- Query type and structure
- Time of day, day of week
- Historical behavior patterns
- Data classification and sensitivity

2.4.3 Query-Level Data Protection

Zero Trust database access enables data protection at the query level, something impossible with network-based or credential-based solutions:

Dynamic Data Masking: Sensitive fields (credit card numbers, social security numbers, personal health information) are automatically masked or redacted based on

the requesting user's clearance level. A developer sees masked data while an authorized analyst sees plaintext—from the same query.

Row-Level Filteringing: The gateway can inject WHERE clauses or modify queries to restrict which rows a user can access, enforcing data boundaries without requiring database-native row-level security configuration.

Column-Level Restrictions: Certain columns can be completely hidden from specific roles, preventing even the awareness of sensitive data's existence.

2.4.4 Complete Auditability and Traceability

Zero Trust database access provides audit logging that captures not just that an action occurred, but the complete context:

- **Who:** Actual human or service identity, not just database username
- **What:** Exact SQL query executed, including results and data accessed
- **When:** Precise timestamp with session context
- **Where:** Source location, network details, client application
- **Why:** Request context, approval workflows if applicable
- **Outcome:** Success, failure, policy denials, data returned

This audit trail is immutable, centralized, and sufficient for forensic investigation, compliance reporting, and threat detection.

2.4.5 What zGate Implements

The zGate architecture embodies these Zero Trust principles through its three-component design:

Gateway Server: Implements protocol handlers for MySQL and MSSQL, intercepting all database traffic at the wire protocol level. The gateway's policy engine evaluates every query against configured rules, the dispatcher manages connection routing, and session managers track user activity in real-time. Users connect to zGate using standard database clients, but the gateway mediates all communication with backend databases.

Command-Line Interface: Provides developers and analysts with a streamlined authentication flow. Users authenticate with their organizational identity, receive time-limited JWT tokens, and establish database sessions without ever handling production credentials. The CLI manages token storage and renewal transparently.

Web Administration Dashboard: Enables security teams to define role-based access control policies, configure database connections, manage user permissions, and

monitor active sessions and query logs. Administrators visualize access patterns, audit historical activity, and respond to security events through a comprehensive management interface.



Figure 2.1: what zGate offers

2.4.6 Operational Advantages

Beyond security improvements, Zero Trust database access delivers operational benefits:

- **Faster Onboarding:** New developers gain database access through role assignment in minutes, not days of credential provisioning
- **Reduced Credential Rotation Burden:** Database passwords change infrequently since users never access them
- **Simplified Compliance:** Centralized policy enforcement and comprehensive audit logs satisfy regulatory requirements
- **Cross-Database Consistency:** Single policy framework applies uniformly across MySQL, PostgreSQL, MSSQL, and other database types
- **Developer Experience:** Legitimate users experience minimal friction—authentication is transparent and access is granted immediately upon authorization

2.4.7 Addressing the Gaps

Zero Trust database access directly addresses every gap in existing solutions:

Limitation	Zero Trust Solution
Perimeter breach enables full access	Gateway enforces identity verification regardless of network position
VPN grants network-wide access	Access is scoped per-database, per-session, per-query
Bastion hosts still expose credentials	Users never possess or see database credentials
Static database permissions	Dynamic policy evaluation per query
PAM credentials can be misused	Tokens are gateway-specific and cannot directly access databases
DAM is reactive only	Real-time policy enforcement prevents unauthorized queries
Shared credentials prevent accountability	Every action is tied to individual user identity

Table 2.1: Comparison of Traditional Limitations vs. Zero Trust Solutions

3. Project Definition

3.1 Project Definition and Scope

zGate is a comprehensive Zero Trust database access gateway platform designed to eliminate credential exposure, enforce identity-based access control, and provide complete auditability for database operations in enterprise environments. The project encompasses the design, implementation, and deployment of a three-tier architecture that intercepts, authenticates, authorizes, and audits all database access in real-time.

3.1.1 System Components

The platform consists of three integrated components working in concert:

Gateway Server (zGate Core)

- **Protocol Handlers:** Native implementation of MySQL and MSSQL wire protocols, enabling transparent protocol-level interception and inspection of database traffic
- **Authentication System:** JWT-based authentication with configurable token expiration, eliminating the need for users to possess database credentials
- **Policy Engine:** Real-time policy evaluation engine that performs fresh permission lookups for every database operation, supporting role-based and custom permission models
- **Dynamic Proxy Manager:** On-demand creation and management of database proxies with automatic port allocation, eliminating static listener configurations
- **Temporary Credential System:** Automated generation of session-specific database users with unique, cryptographically secure credentials that are automatically purged upon session termination
- **Session Management:** Per-user, per-database session tracking with context preservation and graceful cleanup mechanisms

- **API Server:** RESTful API exposing authentication, database listing, connection management, and session control endpoints

Command-Line Interface (zGate-CLI)

- **Secure Authentication Flow:** Interactive login process with secure credential input and token management
- **Token Storage & Management:** Cross-platform secure token storage using OS keyring with encrypted file fallback, automatic token refresh before expiration
- **Database Discovery:** List all databases accessible to the authenticated user based on their assigned roles
- **Connection Management:** Establish database connections through the gateway with automatic credential handling
- **Session Status:** Real-time visibility into active sessions, token expiration, and authentication state
- **Logout & Cleanup:** Proper session termination with server-side revocation and local credential cleanup

Web Administration Dashboard (zGate-WebUI)

Built with Next.js 16, React 19, and Radix UI, the dashboard provides comprehensive management capabilities:

- **Overview Dashboard:** Real-time system metrics, active sessions, and access patterns visualization
- **Database Management:** Configure database connections, credentials, and connection pooling parameters
- **User Management:** Create, modify, and deactivate user accounts with role assignments
- **Administrator Management:** Dedicated administrative user management with elevated privileges
- **Access Control:** Define and manage roles with granular permission specifications including database scoping and privilege levels (read-only, read-write, admin)
- **Active Sessions Monitoring:** View all active database sessions with user context, connection duration, and query activity

- **Query Execution:** Direct query interface with policy enforcement and audit logging
- **Shared Account Pools:** Manage reusable database account credentials for back-end systems
- **Per-User Database Accounts:** Personal database account management for individual user needs
- **Activity Audit:** Comprehensive audit log viewer with filtering, search, and export capabilities
- **Connected Databases:** Real-time status of database connectivity and health

3.1.2 Scope Boundaries

In Scope:

- MySQL, Postgres and MSSQL protocol support with complete authentication and query interception
- Role-based access control with hierarchical permission models
- Temporary database user lifecycle management
- JWT-based authentication with automatic token refresh
- Dynamic proxy allocation and connection pooling
- Comprehensive audit logging of all database operations
- RESTful API for programmatic access
- Cross-platform CLI with secure credential storage
- Web-based administration interface
- Configuration-driven deployment (YAML-based)
- Session-level access control and monitoring

Out of Scope (Current Phase):

- Oracle, and other database protocol support (planned for future phases)
- Real-time query analysis and threat detection algorithms
- Integration with external identity providers (SAML, LDAP, Active Directory)

- Multi-factor authentication (MFA) enforcement
- Database activity replay and forensic analysis tools
- High-availability and clustering capabilities
- Advanced query rewriting and optimization
- Automated compliance reporting generation

3.2 Objectives

The zGate project aims to achieve the following measurable technical and security objectives:

3.2.1 Core Security Objectives

Eliminate Direct Credential Exposure

- Implement JWT-based authentication system where users never handle production database passwords
- Develop automatic temporary database user creation with session-specific, cryptographically secure credentials
- Ensure temporary credentials are purged immediately upon session termination, preventing credential accumulation
- Achieve zero instances of production credentials appearing in user configuration files, environment variables, or CLI history

Enforce Identity-Based Access Control

- Build a role-based access control (RBAC) engine supporting hierarchical roles and custom permissions
- Implement real-time permission evaluation that reflects role changes immediately without requiring user re-authentication
- Support per-database, per-user authorization with multiple privilege levels (read-only, read-write, admin)
- Enable policy decisions based on fresh configuration lookups rather than stale token claims

Implement Protocol-Level Interception

- Develop native protocol handlers for MySQL and MSSQL wire protocols
- Enable transparent database proxy functionality allowing standard client tools (MySQL Workbench, SSMS) to function without modification
- Intercept and route all database traffic through the gateway with sub-100ms latency overhead for typical operations
- Maintain protocol compatibility ensuring 100% of standard database operations work through the proxy

Provide Comprehensive Audit Logging

- Log all authentication attempts with user identity, timestamp, success/failure status, and failure reasons
- Record complete session lifecycle: establishment, duration, database target, and termination circumstances
- Capture connection metadata sufficient for forensic investigation and compliance verification
- Generate structured logs compatible with standard log aggregation and SIEM tools

3.2.2 Operational Objectives

Deliver Multi-Component Integration

- Build RESTful API server exposing authentication, database listing, connection management, and session control
- Develop cross-platform CLI with secure token storage using OS keyring and automatic token refresh
- Create responsive web administration dashboard supporting all management functions
- Ensure seamless data flow between all three components (Gateway, CLI, WebUI)

Enable Centralized Administration

- Provide web-based interface for managing users, roles, database connections, and access policies

- Support real-time monitoring of active sessions with ability to view connection details and terminate sessions
- Allow administrators to modify permissions and configurations without system restarts
- Implement user and admin account separation with appropriate privilege boundaries

Support Multi-Database Environments

- Design modular architecture allowing support for MySQL and MSSQL through unified interfaces
- Enable addition of new database protocols through implementation of standard handler and manager interfaces
- Maintain per-database configuration for credentials, connection parameters, and access policies
- Support simultaneous connections to multiple database instances of different types

Ensure Production-Ready Stability

- Implement graceful error handling, connection cleanup, and resource management
- Support concurrent sessions from multiple users without resource conflicts
- Provide configuration validation and startup checks to prevent misconfiguration
- Enable zero-downtime configuration reloads for non-breaking changes

3.2.3 Academic and Technical Learning Objectives

Demonstrate Applied Security Architecture

- Apply Zero Trust principles specifically to database access management
- Implement secure authentication flows with proper token lifecycle management
- Design authorization systems with clear separation between authentication, policy evaluation, and enforcement
- Practice secure coding including input validation, SQL injection prevention, and secure credential generation

Develop Full-Stack System Integration Skills

- Build high-performance network services in Go with concurrent connection handling
- Implement database wire protocol parsers and connection proxies
- Create modern React-based web interfaces with real-time data updates
- Develop CLI applications with cross-platform compatibility and user experience focus
- Design and document RESTful APIs for inter-component communication

4. Requirements Engineering

4.1 Functional Requirements

4.1.1 FR-1: User Authentication & Authorization

- User login with bcrypt password hashing
- JWT token management (access & refresh tokens)
- Role-based access control (RBAC)
- Custom user permissions

4.1.2 FR-2: Database Connection Management

- Multi-database support (MSSQL, MySQL)
- Dynamic proxy creation with auto port allocation
- Temporary database users with auto-cleanup
- Secure credential generation

4.1.3 FR-3: Policy Engine

- Real-time access control validation
- Permission-based database filtering
- Multi-level permissions (read, write, admin)

4.1.4 FR-4: Command Line Interface

- Auto token refresh
- Database listing and connection
- Session status monitoring

4.1.5 FR-5: Web User Interface

- Database configuration management
- User and role management
- Admin account management
- Active session monitoring
- Shared account pools
- Personal database accounts
- Comprehensive audit trail

4.1.6 FR-6: Protocol Handling

- MSSQL, MySQL and Postgres protocol support with temp user creation

4.1.7 FR-7: Audit & Logging

- Security event logging
- User context in all logs
- Structured logging with configurable levels

4.2 Non-Functional Requirements

4.2.1 NFR-1: Security

- Zero trust architecture principles
- Bcrypt password hashing
- Encrypted connections
- Token-based session security

Key Metrics:

- Access token expiry: 15 minutes
- Refresh token expiry: 7 days
- Session isolation with temporary users

4.2.2 NFR-2: Performance

- Authentication: < 500ms
- Database listing: < 200ms
- Connection establishment: < 2 seconds
- WebUI page loads: < 1 second

4.2.3 NFR-3: Scalability Targets

- 100 concurrent sessions
- 1000 auth requests/minute
- Memory: < 512MB at normal load

4.2.4 NFR-4: Reliability

- 99.5% uptime during business hours
- Graceful error handling
- Data integrity guarantees
- Automatic resource cleanup

4.2.5 NFR-5: Usability

- Unix-style CLI conventions
- Responsive WebUI design
- Helpful error messages
- Comprehensive documentation

4.2.6 NFR-6: Maintainability

- Go best practices
- Externalized YAML configuration
- Structured logging
- Health check endpoints

4.2.7 NFR-7: Portability

- Cross-platform CLI (Linux, macOS, Windows)
- Modern browser support
- Docker deployment support

4.2.8 NFR-8: Compatibility

- MSSQL Server 2016+
- MySQL 5.7+
- API versioning
- Backward compatibility

4.3 Actors & Use Cases

4.3.1 System Actors

Actor descriptions

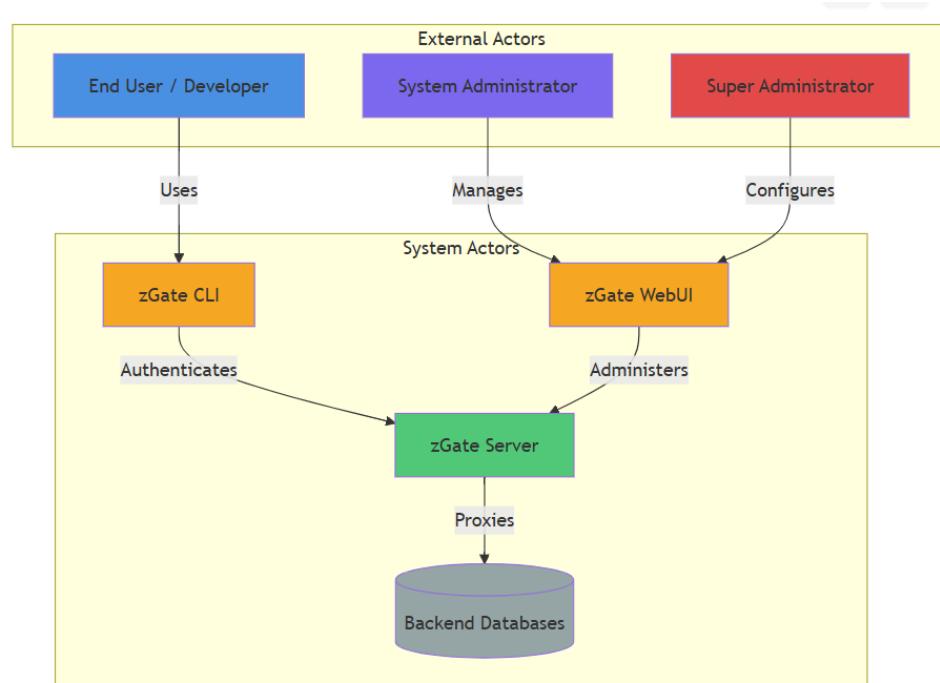


Figure 4.1: Overview of system actors: End Users, System Administrators, Super Administrators, and Backend Databases

End User / Developer

- **Description:** A regular user who needs to access databases for development, analytics, or operations
- **Responsibilities:**
 - Authenticate with the system using CLI
 - Request access to authorized databases
 - Execute database queries within granted permissions
- **Technical Level:** Intermediate (familiar with command line and database clients)

System Administrator

- **Description:** An admin user responsible for day-to-day management of the zGate platform
- **Responsibilities:**
 - Manage user accounts and role assignments
 - Configure database connections
 - Monitor active sessions
 - Review audit logs
 - Manage shared account pools
- **Technical Level:** Advanced (understands database administration and security)

Super Administrator

- **Description:** A privileged admin with full control over the system
- **Responsibilities:**
 - Create and manage admin accounts
 - Configure system-wide settings
 - Manage roles and permissions
 - Handle security incidents
- **Technical Level:** Expert (deep understanding of security and database systems)

zGate Server

- **Description:** The core backend system providing authentication, authorization, and proxy services
- **Responsibilities:**
 - Authenticate users and issue tokens
 - Enforce access policies
 - Create and manage temporary database users
 - Proxy database connections
 - Maintain audit logs

zGate CLI

- **Description:** Command-line client application for end users
- **Responsibilities:**
 - Provide user-friendly interface for authentication
 - Manage token storage and refresh
 - Display available databases
 - Establish database connections

zGate WebUI

- **Description:** Web-based administration dashboard
- **Responsibilities:**
 - Provide graphical interface for system administration
 - Display real-time system status
 - Facilitate configuration management
 - Present audit trail visualization

Backend Databases

- **Description:** Target database systems (MSSQL, MySQL) that users need to access
- **Responsibilities:**
 - Store application data
 - Accept connections from zGate proxy
 - Enforce database-level permissions

4.3.2 Use Case Catalog

UC-1: User Authentication via CLI

Actor: End User

Preconditions: User has valid credentials

Main Flow:

1. User executes `zgate login` command
2. CLI prompts for username and password
3. CLI sends credentials to zGate Server
4. Server validates credentials against user configuration
5. Server generates access and refresh tokens
6. Server returns tokens to CLI
7. CLI stores tokens securely in OS keyring

Postconditions: User is authenticated and tokens are stored

Alternative Flows:

- 3a: Network connection fails → CLI displays error and retries
- 4a: Invalid credentials → Server returns 401, CLI displays error

UC-2: List Available Databases

Actor: End User

Preconditions: User is authenticated

Main Flow:

1. User executes `zgate list` command
2. CLI retrieves stored access token
3. CLI sends list request to zGate Server with token
4. Server validates token signature and expiry
5. Server retrieves user permissions from configuration
6. Server filters database list based on user permissions
7. Server returns list of accessible databases with permission levels

8. CLI displays formatted database list

Postconditions: User sees their accessible databases

Alternative Flows:

- 2a: Token expired → CLI auto-refreshes using refresh token
- 4a: Token invalid → CLI prompts for re-login

UC-3: Connect to Database

Actor: End User

Preconditions: User is authenticated and has permission to target database

Main Flow:

1. User executes `zgate connect --database <db_name>`
2. CLI sends connection request to zGate Server
3. Server validates user has permission for requested database
4. Server retrieves database configuration
5. Server allocates dynamic local port for proxy
6. Server retrieves protocol handler for database type
7. Server creates temporary database user with appropriate permissions
8. Server establishes connection to backend database
9. Server starts proxy listener on allocated port
10. Server returns connection details (host, port, temp credentials) to CLI
11. CLI displays connection information to user
12. User connects using database client with provided credentials

Postconditions: Secure proxy connection established, temporary user created

Alternative Flows:

- 3a: User lacks permission → Server returns 403, CLI displays error
- 4a: Database not found → Server returns 404
- 7a: Backend database unavailable → Server returns error, cleanup any partial state
- 7b: Temporary user creation fails → Server aborts and returns error

UC-4: Auto Disconnect on Session End

Actor: System (zGate Server)

Trigger: User terminates database connection or network fails

Main Flow:

1. Server detects connection closure
2. Server drops temporary database user from backend
3. Server stops proxy listener
4. Server releases allocated port
5. Server logs disconnect event with user context

Postconditions: All session resources cleaned up, temporary user deleted

UC-5: Admin - Add Database Configuration

Actor: System Administrator

Preconditions: Admin is logged into WebUI

Main Flow:

1. Admin navigates to Databases page
2. Admin clicks "Add Database" button
3. WebUI displays database creation form
4. Admin enters database details (name, type, backend address, admin credentials, permissions)
5. WebUI validates form inputs
6. WebUI sends create request to zGate Server API
7. Server validates admin token
8. Server tests connection to backend database
9. Server adds database configuration
10. Server persists configuration to databases.yaml
11. Server returns success response
12. WebUI displays updated database list

Postconditions: New database is available in configuration

Alternative Flows:

- 5a: Validation fails → Display errors on form
- 8a: Backend connection test fails → Return error, don't persist
- 9a: Duplicate database name → Return error

UC-6: Admin - Create User with Roles

Actor: System Administrator

Preconditions: Admin is logged into WebUI, roles exist

Main Flow:

1. Admin navigates to Users page
2. Admin clicks "Create User" button
3. WebUI displays user creation form
4. Admin enters username, password, and selects roles
5. WebUI validates form inputs
6. WebUI sends create user request to zGate Server API
7. Server validates admin token
8. Server hashes password using bcrypt
9. Server adds user configuration with assigned roles
10. Server persists configuration to users.yaml
11. Server returns success response
12. WebUI displays updated user list

Postconditions: New user can authenticate with assigned permissions

Alternative Flows:

- 5a: Validation fails → Display errors
- 9a: Username already exists → Return error

UC-7: Admin - Define Role with Permissions

Actor: System Administrator

Preconditions: Admin is logged into WebUI, databases exist

Main Flow:

1. Admin navigates to Access Control page
2. Admin clicks "Create Role" button
3. WebUI displays role creation form
4. Admin enters role name and description
5. Admin selects databases and permission levels for each
6. WebUI validates inputs
7. WebUI sends create role request to zGate Server API
8. Server validates admin token
9. Server adds role configuration
10. Server persists configuration to roles.yaml
11. Server returns success response
12. WebUI displays updated role list

Postconditions: New role is available for assignment to users

Alternative Flows:

- 6a: Invalid permission level → Display error
- 9a: Role name already exists → Return error

UC-8: Admin - Monitor Active Sessions

Actor: System Administrator

Preconditions: Admin is logged into WebUI

Main Flow:

1. Admin navigates to Sessions page
2. WebUI sends request to fetch active sessions
3. Server validates admin token

4. Server retrieves all active proxy sessions
5. Server returns session details (user, database, start time, connection info)
6. WebUI displays sessions in table format
7. Admin reviews active connections

Postconditions: Admin has visibility into current system usage

Alternative Flows:

- Admin terminates a session:
 1. Admin clicks "Terminate" on a session
 2. WebUI confirms action
 3. Server closes proxy connection
 4. Server deletes temporary database user
 5. WebUI updates session list

UC-9: Admin - Review Audit Logs

Actor: System Administrator

Preconditions: Admin is logged into WebUI

Main Flow:

1. Admin navigates to Activity Audit page
2. Admin optionally applies filters (user, date range, action type)
3. WebUI sends filtered log request to server
4. Server validates admin token
5. Server retrieves matching audit log entries
6. Server returns paginated log data
7. WebUI displays logs in chronological order with search/filter capabilities

Postconditions: Admin can track user activities and security events

UC-10: Auto Token Refresh

Actor: System (CLI)

Trigger: Access token nearing expiry

Main Flow:

1. CLI detects access token will expire within 1 minute
2. CLI retrieves refresh token from secure storage
3. CLI sends refresh request to zGate Server
4. Server validates refresh token
5. Server generates new access token
6. Server returns new access token
7. CLI updates stored access token
8. CLI proceeds with original request

Postconditions: User session continues seamlessly

Alternative Flows:

- 4a: Refresh token expired → CLI prompts user to login again

UC-11: Logout and Token Revocation

Actor: End User

Preconditions: User is authenticated

Main Flow:

1. User executes `zgate logout` command
2. CLI retrieves stored tokens
3. CLI sends logout request to zGate Server with refresh token
4. Server invalidates refresh token
5. Server returns success response
6. CLI deletes tokens from secure storage
7. CLI confirms logout to user

Postconditions: User session is terminated, tokens are invalid

UC-12: Configure Shared Account Pool

Actor: System Administrator

Preconditions: Admin is logged into WebUI, database exists

Main Flow:

1. Admin navigates to Shared Accounts page
2. Admin clicks "Add Shared Account"
3. WebUI displays configuration form
4. Admin selects database, permission level, enters credentials, sets max concurrent users
5. WebUI validates inputs
6. WebUI sends create request to server
7. Server validates admin token
8. Server tests credentials against database
9. Server creates shared account pool configuration
10. Server persists configuration
11. WebUI displays updated shared account list

Postconditions: Shared account pool is available for session allocation

Alternative Flows:

- 8a: Credential test fails → Return error, don't persist

4.4 Use Case Diagrams

4.4.1 End User Case Diagrams

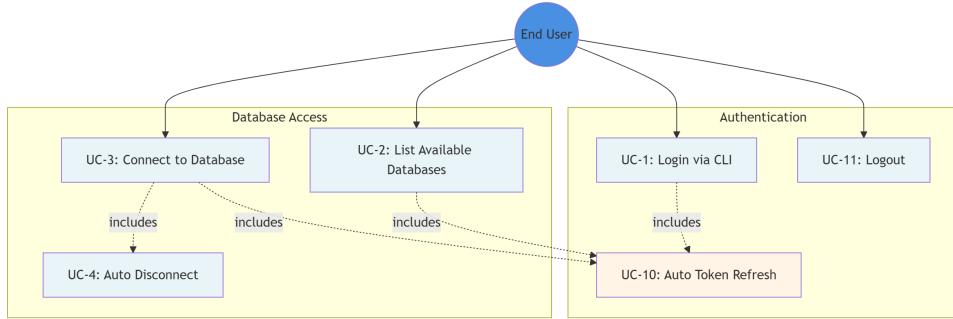


Figure 4.2: End user workflows: authentication, database listing, connection, session management, and logout

4.4.2 Administrator Use Case Diagrams

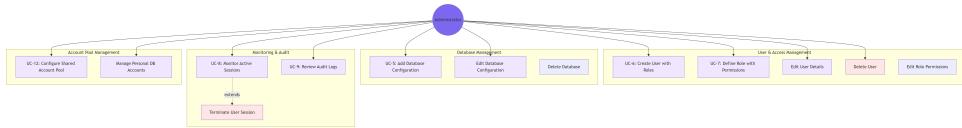


Figure 4.3: Administrator functions: database configuration, user and role management, session monitoring, and audit review

4.4.3 Complete System Use Case Diagrams

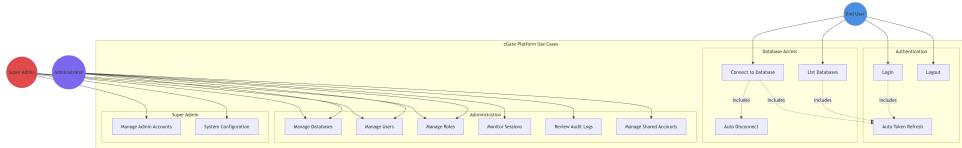


Figure 4.4: Complete system use case diagram showing all actor interactions within the zGate Gateway

4.4.4 System Components Interaction

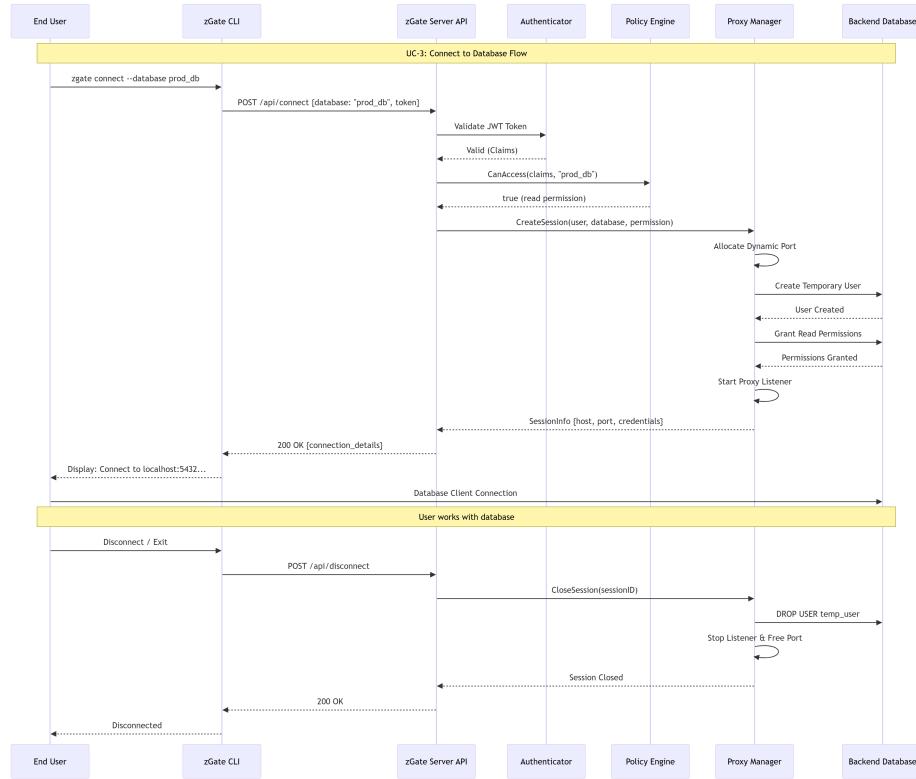


Figure 4.5: System component interactions showing data flow between CLI, Gateway Server, Web Dashboard, Policy Engine, and Backend Databases

4.5 User Stories

4.5.1 End User Stories

Epic 1: Secure Database Access

US-1.1: Login with Credentials

As an end user,

I want to

login using my username and password via the CLI

So that

I can securely authenticate and access authorized databases

Acceptance Criteria:

- CLI prompts for username and password
- Credentials are validated against the server

- Access and refresh tokens are generated on success
- Meaningful error message shown for invalid credentials
- Login completes within 500ms

US-1.2: View My Accessible Databases

As an authenticated user,

I want to

list all databases I have permission to access

So that

I know which databases I can connect to

Acceptance Criteria:

- Command `zgate list` shows my accessible databases
- Each database shows: name, type, my permission level, status, description
- Only databases I have permission for are displayed
- Offline databases are clearly marked
- List loads within 200ms

US-1.3: Connect to a Database

As an authenticated user,

I want to

connect to a specific database using the CLI

So that

I can access data within my permissions

Acceptance Criteria:

- Command `zgate connect --database <name>` initiates connection
- System validates my permission before allowing access
- Temporary credentials are auto-generated
- Connection details (host, port, username, password) are displayed
- I can use any database client with the provided credentials
- I receive clear error if I lack permission
- Connection establishes within 2 seconds

US-1.4: Automatic Session Cleanup

As an end user,

I want

my temporary database access to be automatically cleaned up when I disconnect

So that

I don't have to worry about security hygiene

Acceptance Criteria:

- Temporary database user is deleted when I close my connection
- Proxy resources are released automatically
- Cleanup happens within 30 seconds of disconnect
- No manual intervention required

US-1.5: Seamless Token Refresh

As an authenticated user,

I want

my session to continue without interruption

So that

I don't have to re-login frequently

Acceptance Criteria:

- Access tokens are automatically refreshed before expiry
- I don't experience any interruption during refresh
- I'm only prompted to re-login if my refresh token expires
- Refresh happens transparently in the background

US-1.6: Check Session Status

As an authenticated user,

I want to

check my current session status

So that

I know if I'm logged in and when my session expires

Acceptance Criteria:

- Command `zgate status` shows my current session
- Display includes: username, token expiry, server connection status
- Clear indication if I'm not logged in

US-1.7: Secure Logout

As an authenticated user,

I want to

logout and revoke my session

So that

my credentials are invalidated when I'm done

Acceptance Criteria:

- Command `zgate logout` terminates my session
- Tokens are revoked on the server
- Local token storage is cleared
- Confirmation message is displayed

4.5.2 Administrator Stories

Epic 2: User & Access Management

US-2.1: Create User Accounts

As a system administrator,

I want to

create new user accounts with assigned roles

So that

team members can access databases based on their responsibilities

Acceptance Criteria:

- I can access user creation form in WebUI
- I can enter username and initial password
- I can assign one or more roles to the user
- Password is securely hashed before storage

- User can login immediately after creation
- Duplicate usernames are prevented with clear error

US-2.2: Define Roles with Granular Permissions

As a system administrator,

I want to

define roles with specific database permissions

So that

I can implement least-privilege access control

Acceptance Criteria:

- I can create named roles (e.g., "data_analyst", "developer")
- For each role, I can specify permissions per database
- Permission levels include: read, write, admin
- Roles can be assigned to multiple users
- Changing role permissions affects all assigned users immediately

US-2.3: Manage User-Role Assignments

As a system administrator,

I want to

add or remove roles from existing users

So that

I can adjust access as team responsibilities change

Acceptance Criteria:

- I can view current role assignments for any user
- I can add new roles to a user
- I can remove roles from a user
- Changes take effect immediately for new connections
- Audit log captures all role changes

Epic 3: Database Configuration

US-3.1: Add Database Configurations

As a system administrator,

I want to

add new database connections to the system

So that

users can access additional data sources

Acceptance Criteria:

- I can provide: name, type (MSSQL/MySQL), backend address, admin credentials
- I can specify available permission levels for the database
- Connection is tested before saving
- Configuration is persisted to databases.yaml
- Database appears in user lists immediately
- Clear error shown if connection test fails

US-3.2: Edit Database Settings

As a system administrator,

I want to

update existing database configurations

So that

I can reflect infrastructure changes

Acceptance Criteria:

- I can modify connection details (address, credentials)
- I can update available permissions
- Changes are validated before saving
- Active sessions using old config are not disrupted
- New connections use updated configuration

US-3.3: Remove Decommissioned Databases

As a system administrator,

I want to

delete database configurations that are no longer needed

So that

users don't see outdated options

Acceptance Criteria:

- I can delete a database configuration
- System confirms deletion to prevent accidents
- Database is immediately removed from user lists
- Deletion is logged in audit trail

Epic 4: Monitoring & Audit

US-4.1: Monitor Active Sessions

As a system administrator,

I want to

see all active database sessions in real-time

So that

I can monitor system usage and detect anomalies

Acceptance Criteria:

- I can view list of all active connections
- Each session shows: user, database, connection time, status
- List updates in real-time or on refresh
- I can see connection details (proxy port, temp username)

US-4.2: Terminate Suspicious Sessions

As a system administrator,

I want to

forcefully terminate active sessions

So that

I can respond to security incidents

Acceptance Criteria:

- I can select any active session and terminate it
- System confirms action before executing
- Session proxy is closed immediately
- Temporary database user is deleted
- Action is logged in audit trail
- User receives connection closed error

US-4.3: Review Comprehensive Audit Logs

As a system administrator,

I want to

review all user activities and system events

So that

I can investigate issues and maintain compliance

Acceptance Criteria:

- I can view chronological audit log in WebUI
- Each entry includes: timestamp, user, action, outcome, context
- I can filter by: user, date range, action type, success/failure
- I can search logs by keywords
- Logs include: logins, connection requests, admin actions, errors
- Logs are paginated for performance

US-4.4: Export Audit Logs

As a system administrator,

I want to

export audit logs for compliance reporting

So that

I can provide evidence of access controls

Acceptance Criteria:

- I can export filtered logs to CSV or JSON

- Export includes all relevant fields
- Large exports don't timeout or crash

Epic 5: Advanced Access Control

US-5.1: Configure Shared Account Pools

As a system administrator,

I want to

create pools of shared database credentials

So that

multiple users can share backend accounts efficiently

Acceptance Criteria:

- I can create shared account pool for a database
- I can specify: database, permission level, credentials, max concurrent users
- System allocates shared account when user connects
- System prevents exceeding max concurrent limit
- User transparently gets shared credentials
- Account is returned to pool on disconnect

US-5.2: Assign Personal Database Accounts

As a system administrator,

I want to

assign personal database accounts to specific users

So that

some users have dedicated credentials for auditing

Acceptance Criteria:

- I can create personal account for user-database pair
- I can provide specific database credentials
- User always gets their personal account when connecting
- Personal accounts override shared pools
- I can revoke personal accounts

US-5.3: Define Custom User Permissions

As a system administrator,

I want to

grant custom permissions to individual users beyond their roles

So that

I can handle special cases without creating new roles

Acceptance Criteria:

- I can add database permissions directly to a user
- Custom permissions combine with role-based permissions
- I can see both role and custom permissions in user view
- Custom permissions can be removed independently

4.5.3 Super Administrator Stories

Epic 6: Platform Administration

US-6.1: Manage Admin Accounts

As a super administrator,

I want to

create and manage admin user accounts

So that

I can grant administrative access to trusted personnel

Acceptance Criteria:

- I can create new admin accounts with secure passwords
- I can view all existing admin accounts
- I can disable or delete admin accounts
- Regular users cannot access admin functions
- Admin account changes are logged

US-6.2: Configure System Settings

As a super administrator,

I want to

configure system-wide settings

So that

I can tune the platform for our environment

Acceptance Criteria:

- I can set token expiry durations (access and refresh)
- I can configure backend connection timeouts
- I can set log levels (DEBUG, INFO, WARN, ERROR)
- Settings are validated before applying
- Configuration changes are persisted

US-6.3: Perform System Health Checks

As a super administrator,

I want to

check the health of all system components

So that

I can proactively identify issues

Acceptance Criteria:

- I can view status of: API server, proxy manager, backend databases
- Each database shows: online/offline status, last check time
- I can manually trigger connectivity tests
- Failed health checks generate alerts

4.5.4 System Stories (Non-Interactive)

Epic 7: Automated Operations

US-7.1: Automatic Resource Cleanup

As the zGate system,

I need to

automatically clean up temporary resources

So that

database systems don't accumulate orphaned users

Acceptance Criteria:

- Temporary database users are deleted within 30s of disconnect
- Proxy listeners are stopped immediately on session close
- Allocated ports are freed for reuse
- Cleanup happens even if client disconnects ungracefully
- Failed cleanup attempts are retried with exponential backoff
- Persistent cleanup failures are logged as errors

US-7.2: Real-Time Permission Enforcement

As the zGate system,

I need to

evaluate permissions at request time using current configuration

So that

permission changes take effect immediately

Acceptance Criteria:

- Each connection request checks latest roles and permissions
- Configuration file changes are detected and reloaded
- Users with revoked permissions cannot establish new connections
- Permission checks complete within 50ms
- Permission evaluation is thread-safe

US-7.3: Graceful Error Handling

As the zGate system,

I need to

handle errors gracefully without leaking sensitive information

So that

users get helpful feedback while maintaining security

Acceptance Criteria:

- User-facing errors don't expose internal paths or stack traces
- Authentication failures return generic "invalid credentials" message
- Network errors suggest retry with backoff
- All errors are logged with full context
- Critical errors trigger alerts to administrators

US-7.4: Comprehensive Audit Logging

As the zGate system,

I need to

log all security-relevant events

So that

administrators can audit access and investigate incidents

Acceptance Criteria:

- All authentication attempts are logged (success and failure)
- All database access requests are logged with user context
- All administrative actions are logged
- Logs include: timestamp, user, action, outcome, IP address, session ID
- Logs are structured (JSON format option)
- Log rotation prevents disk exhaustion
- Sensitive data (passwords, credentials) are never logged

5. Proposed Solution

5.1 Solution Overview

The proposed solution is **zGate**, a comprehensive Zero-Trust Database Access Platform that fundamentally changes how organizations manage database security and access control. Rather than relying on shared credentials and static permissions, zGate introduces a dynamic, policy-driven approach where temporary credentials are created for each user session, eliminating the risks associated with credential sprawl and unauthorized access.

5.1.1 The Core Problem

Traditional database access models suffer from several critical security and operational challenges:

1. **Shared Credentials:** Multiple users share common database accounts, making it impossible to audit who performed which actions
2. **Static Permissions:** Once granted, permissions remain indefinitely, increasing the attack surface
3. **Credential Sprawl:** Database passwords stored in configuration files, wikis, and password managers
4. **Lack of Visibility:** No centralized view of who has access to which databases
5. **Manual Overhead:** Granting and revoking access requires manual database administration
6. **Compliance Gaps:** Difficult to prove least-privilege access and maintain audit trails

5.1.2 The zGate Solution

zGate addresses these challenges through a multi-layered architecture built on three core principles:

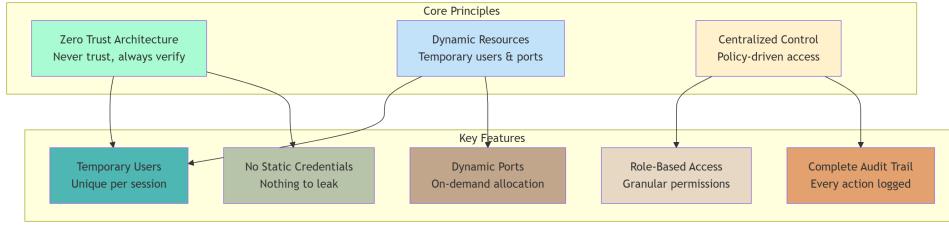


Figure 5.1: zgate proposed solution architecture

5.2 Core Components

5.2.1 zGate Server (Backend)

The server is the heart of the platform, handling all security, policy enforcement, and proxy operations.

Backend Layers

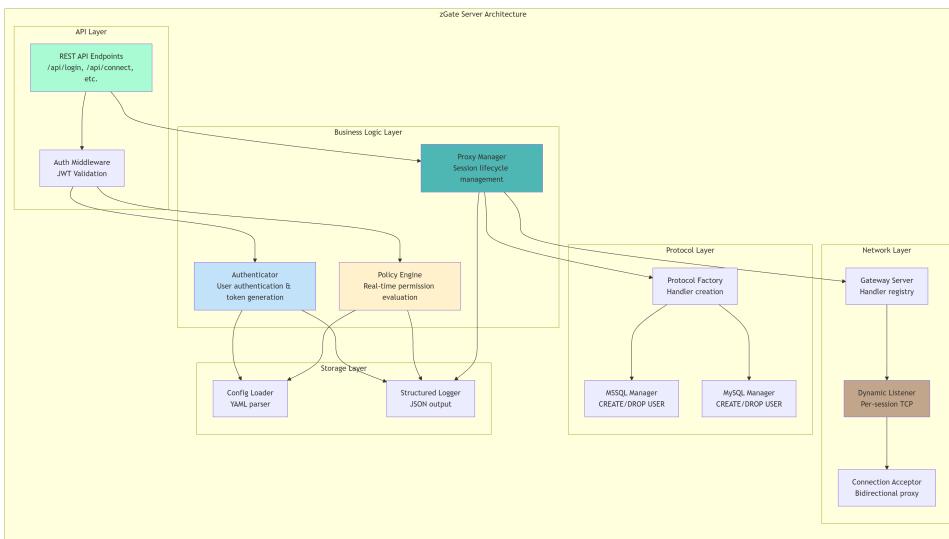


Figure 5.2: zGate backend architecture showing authentication, policy engine, and proxy layers

Key Features

1. JWT-Based Authentication

- Access tokens (15-minute TTL) for API requests
- Refresh tokens (7-day TTL) for session persistence
- Secure token generation with HMAC-SHA256 signatures

- Automatic token refresh in CLI

2. Real-Time Policy Enforcement

- Permissions evaluated from current configuration on every request
- Role-based access control (RBAC) with support for multiple roles per user
- Custom permissions at user level for exceptions
- Immediate effect when permissions are changed (no cache invalidation needed)

3. Dynamic Proxy Management

- On-demand creation of proxy sessions
- Automatic port allocation from ephemeral range
- Temporary database user creation with appropriate grants
- Automatic cleanup on disconnect or crash recovery

4. Protocol Abstraction

- Pluggable architecture for database types
- Current support: MSSQL, MySQL
- Easy extensibility for PostgreSQL, Oracle, etc.
- Database-specific SQL for user management

5. Comprehensive Logging

- Structured logging with key-value pairs
- Every security event logged with user context
- Support for JSON output for log aggregation
- Configurable log levels (DEBUG, INFO, WARN, ERROR)

5.2.2 zGate CLI (Client)

The command-line interface provides developers with a simple, secure way to access databases.

CLI Workflow

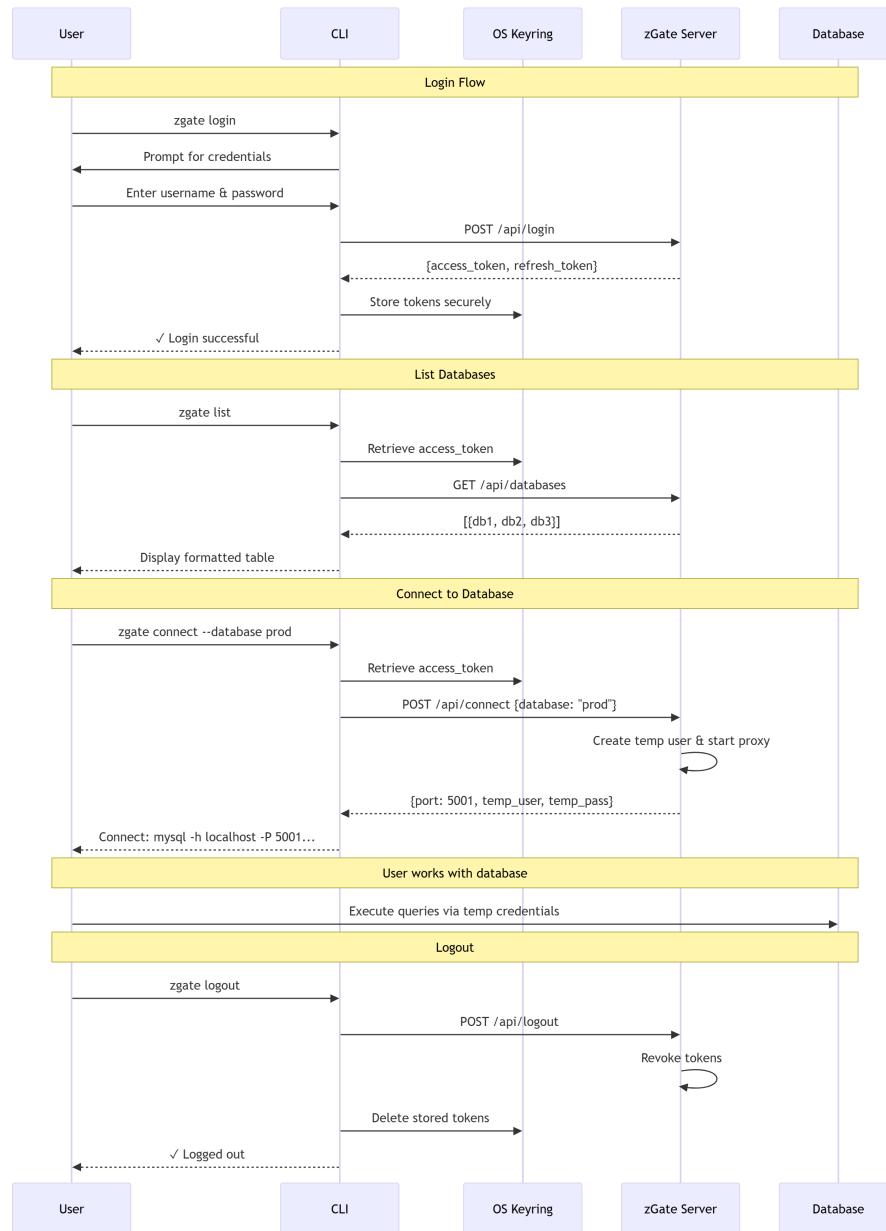


Figure 5.3: zGate CLI workflow illustrating token management and database access process

CLI Features

1. Secure Token Storage

- OS-native keyring integration (Keychain on macOS, Credential Manager on Windows, Secret Service on Linux)
- Encrypted file fallback for headless environments
- Never stores plaintext credentials

2. Automatic Token Refresh

- Detects token expiry proactively
- Refreshes access token using refresh token
- Seamless user experience without re-authentication

3. User-Friendly Interface

- Simple, intuitive commands following Unix conventions
- Colored output for better readability
- Helpful error messages with troubleshooting hints
- Progress indicators for long operations

4. Cross-Platform Support

- Single binary for Windows, macOS, Linux
- No runtime dependencies
- Portable and lightweight (~10MB)

5.2.3 zGate WebUI (Admin Dashboard)

The web interface provides administrators with a comprehensive control panel for managing the entire platform.

Dashboard Architecture

Admin Features

1. Database Management

- Add/edit/delete database configurations
- Test database connectivity before saving
- View all configured databases with status
- Manage available permissions per database

2. User & Role Management

- Create users with bcrypt-hashed passwords

- Assign multiple roles to users
- Define roles with database-specific permissions
- Add custom permissions to individual users

3. Session Monitoring

- Real-time view of active database connections
- See: user, database, port, temporary username, duration
- Terminate sessions if needed (security incidents)
- Filter and search sessions

4. Audit Trail

- Comprehensive logs of all user activities
- Filter by user, date range, action type
- Search within log entries
- Export for compliance reporting

5. Advanced Features

- Shared account pools for efficient credential management
- Personal database accounts for specific users
- Query execution interface (future)
- Connected databases view

5.3 Key Technologies & Design Decisions

Technology Stack

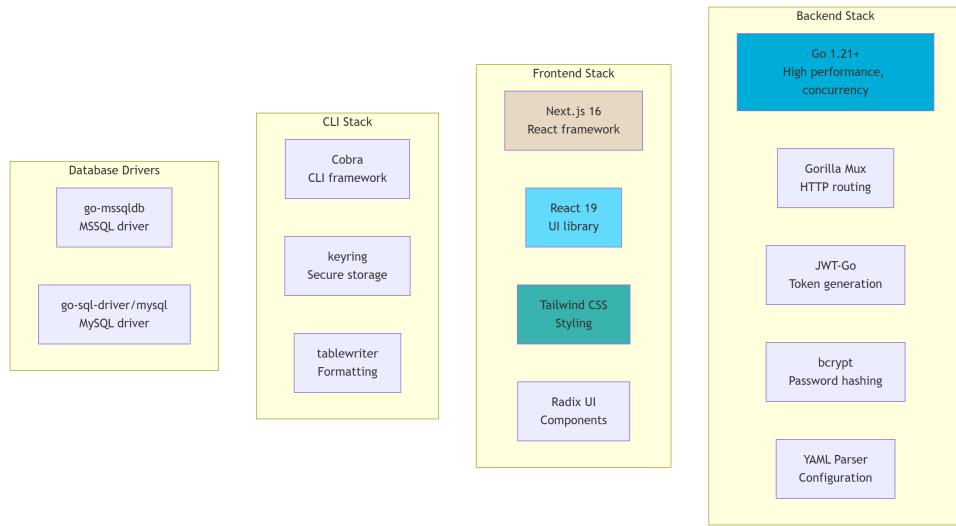


Figure 5.4: zGate technology stack

5.4 Security Architecture

Security is woven into every layer of the zGate platform through multiple defense mechanisms.

5.4.1 Security Layers

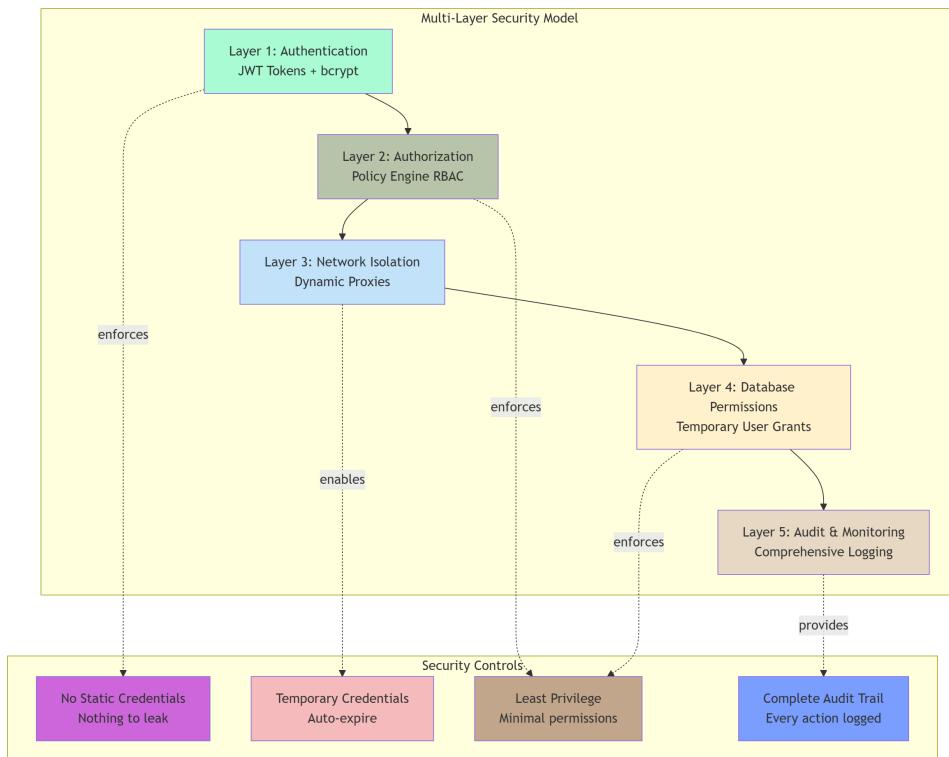


Figure 5.5: Multi-layered security architecture in zGate platform

5.4.2 Security Guarantees

Threat	Mitigation	Implementation
Credential Theft	No static credentials exposed to users	Temporary users with auto-generated passwords
Privilege Escalation	Database-level permission enforcement	Grants limited to user's role (read/write/admin)
Unauthorized Access	Real-time policy evaluation	Every connection request checked against current config
Session Hijacking	Short-lived JWT tokens	15-minute access token expiry
Password Cracking	Strong password hashing	bcrypt with work factor 10
Audit Evasion	Mandatory logging	Every API call, connection, disconnect logged
Resource Exhaustion	Automatic cleanup	Temporary users deleted within 30 seconds of disconnect
Man-in-the-Middle	Encrypted connections	TLS support for backend database connections

Table 5.1: Security threats and their mitigations in zGate

5.5 Advantages of the Proposed Solution

Key Benefits

Enhanced Security

- Eliminates shared credentials
- Temporary users auto-deleted
- No credential sprawl
- Cryptographically secure random passwords
- Short-lived JWT tokens
- Database-level permission enforcement

Operational Efficiency

- Automated user provisioning
- Self-service for developers (via CLI)
- Centralized access management
- No manual database administration
- Quick onboarding/offboarding

Compliance & Audit

- Complete audit trail
- User-attributed actions
- Provable least-privilege
- Real-time access reporting
- Exportable logs

Developer Experience

- Simple CLI commands
- Works with existing tools
- No VPN/bastion required
- Automatic token refresh
- Cross-platform support

Scalability

- Handles thousands of concurrent sessions
- Stateless API (horizontal scaling)
- Lightweight memory footprint
- Dynamic resource allocation

6. Alignment with International Standards

6.1 PCI DSS

- **Access Control (Req. 7):** Role-based access with least privilege per database
- **Authentication (Req. 8):** Unique user identification, no shared credentials, time-limited sessions
- **Audit Logging (Req. 10):** Complete tracking of all database access with user identity, timestamp, and session details

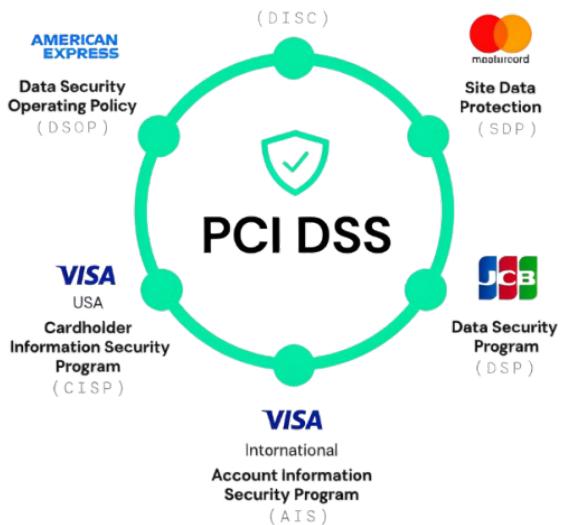


Figure 6.1: zGate alignment with PCI DSS requirements for access control, authentication, and audit logging

6.2 HIPAA

- **Access Control:** Unique user authentication and role-based database access to ePHI
- **Audit Controls:** Comprehensive logs of who accessed what database and when
- **Authentication:** JWT-based identity verification before database access
- **Automatic Log-off:** Session expiration and token timeout mechanisms

6.3 GDPR

- **Data Security (Art. 32):** Identity-based authentication, authorization, and encrypted credentials
- **Access Control (Art. 5):** Role-based permissions limiting data access scope
- **Audit Records (Art. 30):** Complete logging of processing activities with user context
- **Breach Detection (Art. 33):** Real-time monitoring and immediate session termination capabilities



Figure 6.2: zGate alignment with GDPR requirements for data security, access control, audit records, and breach detection

6.4 ISO 27001

- **User Access Management (A.9.2):** Formal registration, role assignment, and access revocation
- **Access Control (A.9.4):** Secure authentication, temporary credentials, session timeouts
- **Logging & Monitoring (A.12.4):** Event logging for all activities with protected audit trails
- **Compliance Reviews (A.18):** Audit logs and monitoring dashboard for compliance verification

7. Competitor & Market Analysis

7.1 Competitor Analysis

7.1.1 Comparison Table

7.1.2 What Competitors Lack

7.2 Market Research

7.2.1 Market Overview

7.2.2 Zero Trust Demand

7.2.3 Market Challenges & Needs

7.2.4 Regulatory Drivers

7.2.5 Trends & Opportunities

7.2.6 Landscape Summary

8. Scientific Research & Literature Review

8.1 Generative AI-Enhanced Cybersecurity Framework for Enterprise Data Privacy Management

8.1.1 Purpose of the Study

The paper addresses the growing need for organizations to secure sensitive enterprise data (e.g., financial transactions, patient records, IoT data) while still enabling advanced detection of cyber threats. Traditional security controls and anomaly detection methods often:

- Miss new/unknown attack patterns
- Require direct use of real sensitive data, creating privacy and compliance risks

Goal: The authors propose a Generative AI-enhanced framework that combines Generative Adversarial Networks (GANs), Variational Autoencoders (VAEs), and traditional machine learning (ML/DL) anomaly detection with strong privacy-preserving methods (Differential Privacy, encryption, masking). The aim is to balance data privacy, detection accuracy, and computational efficiency.

8.1.2 Framework Overview

The proposed framework works as an end-to-end pipeline with the following main components:

- **Data ingestion:** Collect logs (network, system, application) via tools like Splunk/ELK
- **Generative AI layer (GANs & VAEs):** Create synthetic, privacy-safe data that mimic real-world patterns without exposing identities
- **Privacy layer:** Apply Differential Privacy ($\varepsilon = 0.1$ for highly sensitive data), AES-256 encryption, TLS 1.3, and masking
- **Anomaly detection engine:** Train models like Random Forest, SVM, and LSTM on synthetic + sanitized data to detect unusual activity

- **Monitoring & alerting:** Real-time detection with dashboards and alert systems

Analogy: Instead of training guards with real customer data (which is risky), the system uses highly realistic "actors" (synthetic data) to train them — ensuring the guards learn effectively without ever seeing the real people.

8.1.3 Implementation & Experiments

The framework was tested in three simulated enterprise domains:

- **Finance (transaction logs):** 94% accuracy, 95% recall, $\sim 1.2\text{--}1.5$ seconds per transaction
- **Healthcare (EHR access logs):** 96% accuracy, 93% precision, ~ 1.5 seconds per event
- **Smart City/IoT (sensor data):** 91% accuracy, $F1 \approx 90\%$, latency $\downarrow 100$ ms at the edge

Performance trade-offs:

- GAN framework: $\sim 96\%$ accuracy, moderate compute (4GB GPU, 2.5h training)
- LSTM: $\sim 97\%$ accuracy but higher GPU needs (6GB)
- Traditional ML (RF/SVM): lower accuracy ($\sim 92\text{--}94\%$) but lighter
- Very high-accuracy CNN ($>99\%$): impractical resource usage

8.1.4 Privacy & Security Features

- **Differential Privacy:** Adds noise to hide individual user data, ensuring compliance with GDPR/HIPAA
- **Encryption:** AES-256 for data at rest, TLS 1.3 for data in transit
- **Access control:** Role-based restrictions
- **Data masking:** Obscures identifiers in logs

8.1.5 Contributions to the Paper

- First comprehensive framework integrating Generative AI + privacy techniques + anomaly detection
- Provides balanced performance: strong accuracy without extreme computational demands
- Applicable across finance, healthcare, and IoT
- Offers implementation guidance with practical tools (TensorFlow, PyTorch, Scikit-learn, PySyft)

8.1.6 Advantages & Limitations

Advantages:

- Protects privacy while enabling effective training
- Can detect novel/rare attacks better by augmenting datasets with synthetic samples
- Works across domains, modular and adaptable
- More resource-efficient than some deep CNN methods

Limitations:

- Results are simulated, not from live production environments
- Quality of synthetic data can affect detection accuracy
- Managing GANs, VAEs, DP, and anomaly detectors is operationally complex
- Differential Privacy trade-off: stronger privacy (smaller ε) may reduce model accuracy

8.1.7 Relevance to Our Project

This study is directly relevant because:

- Our project focuses on secure access and monitoring of sensitive databases
- The paper's synthetic-data + anomaly detection pipeline is a practical approach to train models without exposing real database queries/records
- Techniques like Differential Privacy, RBAC, AES-256 encryption overlap with Zero Trust principles (least privilege, continuous monitoring, encryption everywhere)
- Their results show that real-time detection with privacy is feasible, which strengthens the foundation for our Zero Trust access model

8.2 The Significance of Artificial Intelligence in Zero Trust Technologies: A Comprehensive Review

8.2.1 Problem Addressed

Traditional security models assume anything inside a company's network can be trusted. With today's cloud, remote work, and hybrid environments, this assumption no longer holds. Attackers exploit cloud resources, lateral movement inside networks, and slow manual controls. The study addresses how Artificial Intelligence (AI) can enhance the Zero Trust (ZT) model to meet these modern challenges.

8.2.2 Methodology

- Literature review (20+ studies examined)
- Synthesized how AI is applied across ZT building blocks (IAM, MFA, EDR, ZTNA, SASE, Network Analytics)

8.2.3 Key Contributions of AI to Zero Trust

Identity & Access Management (IAM)

- **Authentication:** Adaptive and continuous (AI monitors typing, device, location; flags anomalies)
- **Authorization:** Intelligent Role-Based Access Control (AI suggests roles, prevents "over-privilege")
- **Administration:** Automated onboarding/offboarding, policy adjustments
- **Audit/Compliance:** AI generates audit trails, suggests policies, detects compliance gaps

Adaptive Multi-Factor Authentication (AMFA)

- AI adjusts authentication strength based on risk (low → password; medium → OTP; high → biometrics)
- Balances usability with security

Endpoint Detection & Response (EDR)

- AI baselines device behavior, detects anomalies, reduces false positives
- Automates containment (isolate compromised laptops)

Zero Trust Network Access (ZTNA) & Secure Access Service Edge (SASE)

- ZTNA grants application-level access (not full network like VPN)
- SASE combines SD-WAN + ZTNA + CASB + FWaaS; AI analyzes telemetry, recommends segmentation
- AI enables dynamic microsegmentation and automated policy creation

8.2.4 Findings

- AI strengthens Zero Trust by making it continuous, adaptive, and automated
- AI reduces human error, speeds up detection, and scales across large organizations
- AI integration is critical for modern cloud and hybrid infrastructures

8.2.5 Comparison with Traditional Methods

- Traditional perimeter security = trust anyone inside
- Zero Trust with AI = checkpoint at every request making context-based decisions in real time

8.2.6 Relevance to Our Project

- IAM insights → directly applicable for database user role mining & continuous verification
- Adaptive MFA → useful for database login protection
- EDR concepts → extend to database clients/endpoints
- ZTNA & SASE → inspire database-level microsegmentation (grant per-query or per-app access)
- Network analytics → parallels database traffic analysis for anomaly detection

8.3 Securing Digital Identity in the Zero Trust Architecture: A Blockchain Approach to Privacy-Focused Multi-Factor Authentication

8.3.1 Problem Addressed

- Traditional MFA depends on centralized servers, which are vulnerable to outages and breaches
- Zero Trust architectures require continuous, strong identity verification, but current MFA approaches are limited in resilience and privacy

The study addresses these issues by designing a decentralized, privacy-focused MFA mechanism that eliminates single points of failure and ensures secrets remain private.

8.3.2 Research Goals

- Build a decentralized authentication system aligned with Zero Trust principles
- Ensure privacy-preserving verification so users never expose OTPs
- Provide auditability and traceability of authentication events
- Deliver a realistic proof-of-concept that demonstrates feasibility and performance

8.3.3 Proposed System

- **Distributed Authentication Mechanism (DAM):** validator nodes collectively handle authentication
- **Distributed OTP Generation:** each validator contributes a random partial secret; combined into full OTP
- **Privacy via zk-SNARKs:** users prove they know the OTP without revealing it
- **Authentication Token:** successful proof leads to issuance of a non-transferable NFT (digital badge) valid for a period

8.3.4 Experimental Results

- **Performance:** comparable to real-world MFA timings (~20 seconds average)
- **Security:** analysis shows probability of attack success is negligible due to distribution, cryptographic verification, and non-transferability

8.3.5 Key Findings and Contributions

- Decentralized authentication reduces single points of failure
- OTP secrets are never exposed; verification occurs through zk-SNARK proofs
- Immutable, auditable on-chain logs improve accountability
- Non-transferable NFTs prevent token theft or resale

8.3.6 Real-World Applications

- **Banking & Fintech:** customers receive distributed OTPs and prove knowledge via zk-SNARK, receiving session tokens for access
- **Corporate IT (Zero Trust):** employees authenticate and receive short-lived NFTs, ensuring continuous verification without exposing passwords
- **Developer Platforms:** integration with existing blockchain and web authentication frameworks for higher resilience

8.3.7 Relevance to Our Project

This research is directly relevant because it shows how decentralized, privacy-preserving multi-factor authentication can be integrated into a Zero Trust architecture; a blueprint for secure identity verification that we can adapt for controlling database access in our Zero Trust Gateway.

8.3.8 Conclusion

This research provides a comprehensive, innovative, and feasible approach to MFA in Zero Trust environments. By using distributed OTP generation, zk-SNARK verification, and non-transferable authentication tokens, it resolves critical weaknesses in traditional MFA. While some limitations exist (cost, setup trust, validator reputation), the overall contribution strongly supports the feasibility of implementing privacy-focused, decentralized identity verification in real-world systems.

8.4 Paper 4

8.5 Paper 5

8.6 Paper 6

8.7 Paper 7

8.8 Research References

- **Paper 1:** Generative AI-Enhanced Cybersecurity Framework for Enterprise Data Privacy Management
<https://www.mdpi.com/2073-431X/14/2/55>
- **Paper 2:** The Significance of Artificial Intelligence in Zero Trust Technologies: A Comprehensive Review
<https://link.springer.com/article/10.1186/s43067-024-00155-z>
- **Paper 3:** Securing Digital Identity in the Zero Trust Architecture: A Blockchain Approach to Privacy-Focused Multi-Factor Authentication
<https://ieeexplore.ieee.org/abstract/document/10505915>

9. Technical Background

10. System Architecture

10.1 High-Level Architecture Diagram

10.2 Main System Components

10.3 Component Communication Flow

10.4 Tech Stack Summary

11. Detailed Architecture of the Proxy

Overview

The zGate proxy is the core component that provides secure, zero-trust database access through dynamic port allocation and temporary user management. It acts as an intelligent intermediary between users and backend databases, enforcing policy, managing credentials, and monitoring all database traffic.

Key Components

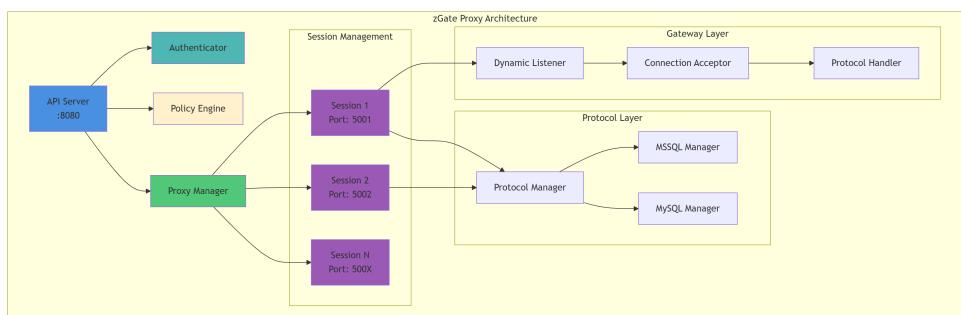


Figure 11.1: system components

11.1 Connection Lifecycle

The connection lifecycle encompasses the complete journey from user authentication to database disconnection, including all intermediate states and transitions.

11.1.1 Lifecycle States

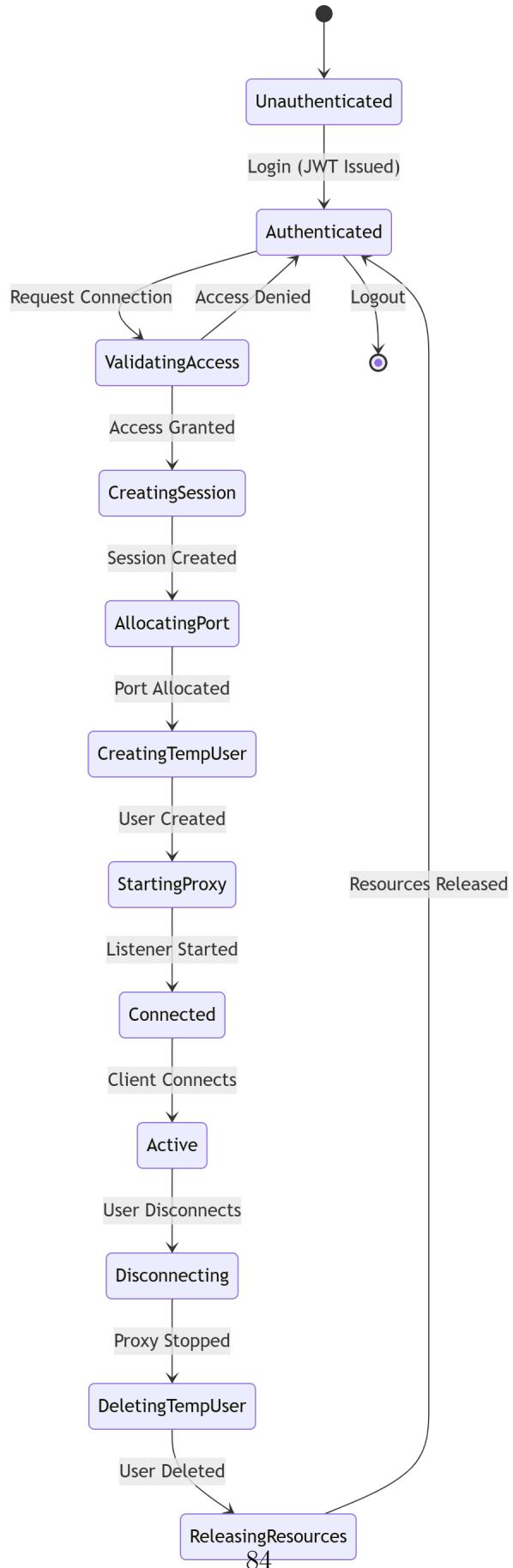


Figure 11.2: connection lifecycle states

11.1.2 Detailed Lifecycle Flow

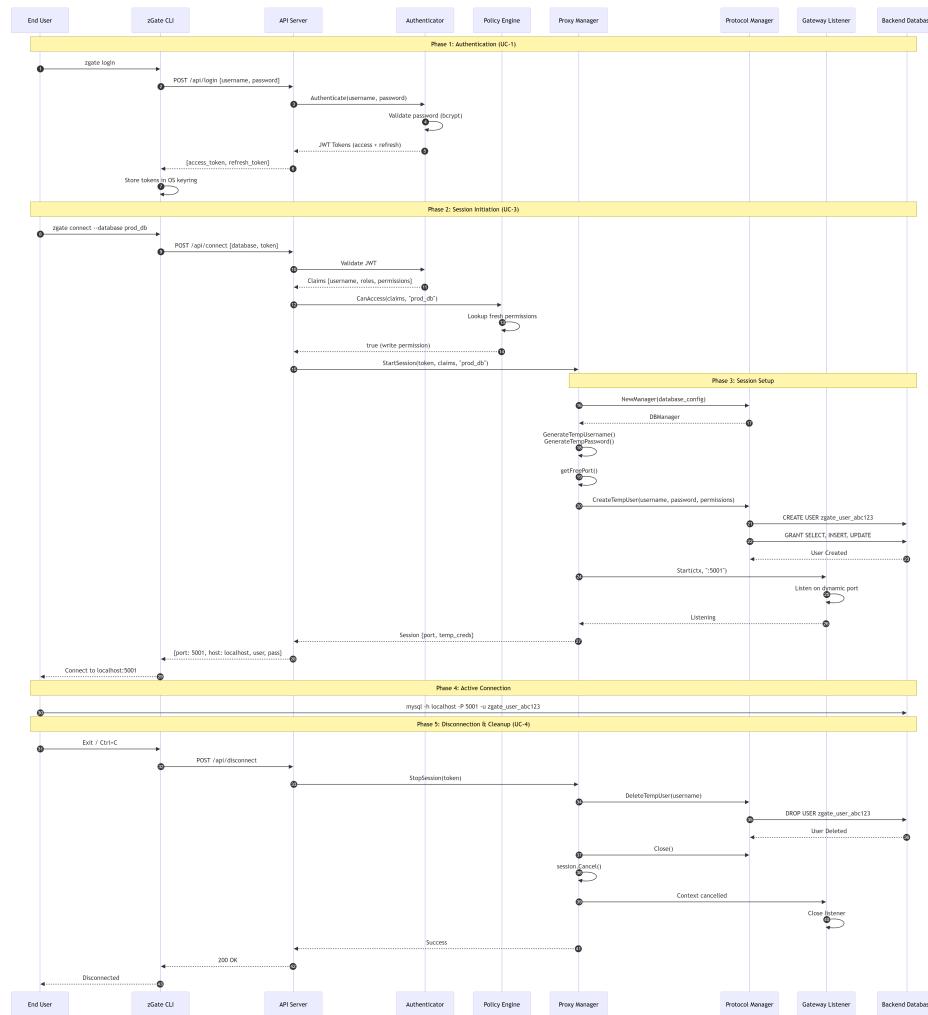


Figure 11.3: Detailed connection lifecycle flow

11.2 Authentication Flow

Authentication is a multi-layered process involving credential validation, token generation, and permission resolution.

11.2.1 Authentication Architecture

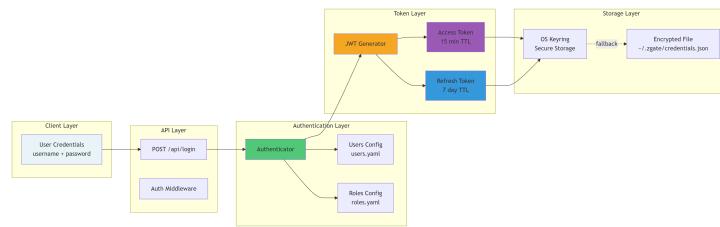


Figure 11.4: Authentication architecture

11.2.2 Token Structure

Access Token Claims

```
{  
    key "sub": val "kemo@company.com",  
    key "username": val "kemo@company.com",  
    key "roles": [val "data_analyst"],  
    key "permissions":  
        [key "database": val "azure\mssql\prod", key "level": val "read"],
```

Refresh Token Claims

```
{  
    key "sub": val "kemo@company.com",  
    key "username": val "kemo@company.com",  
    key "iat": 1702345678, key "exp": 1702950478,  
    key "type": val "refresh"  
}
```

11.2.3 Token Refresh Flow

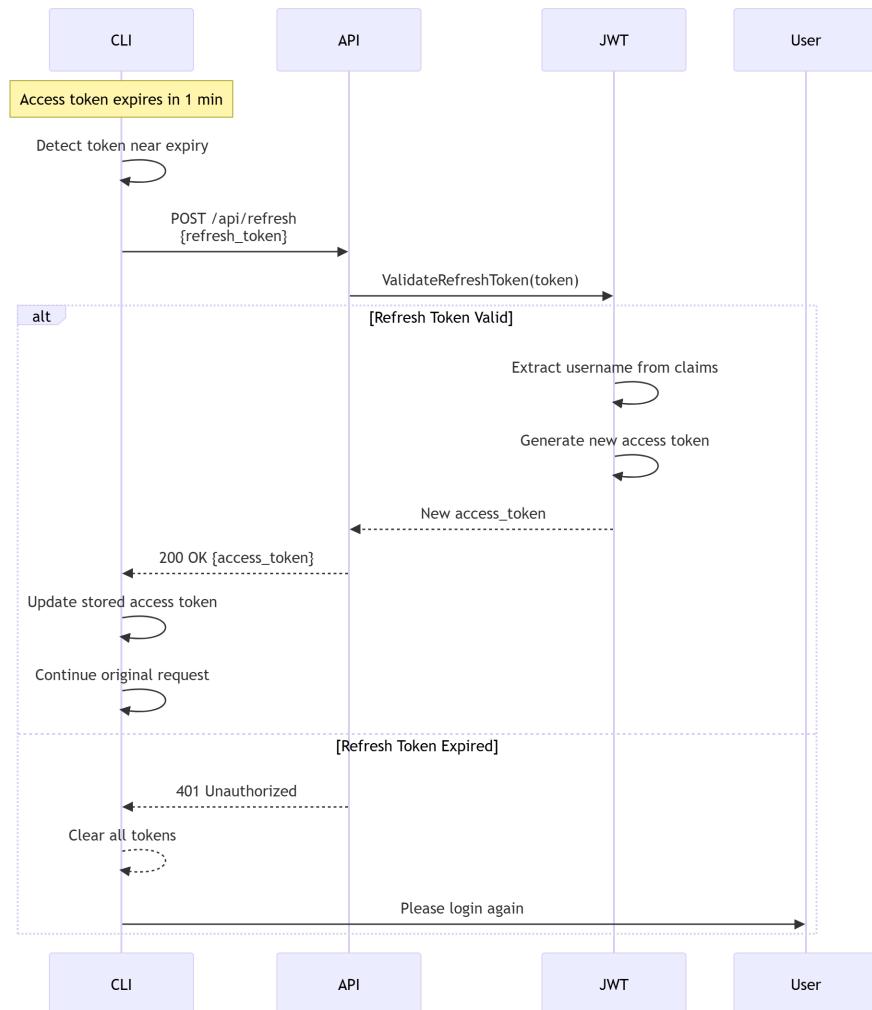


Figure 11.5: Token refresh flow

11.3 Query Filtering Flow

Query filtering ensures that users can only execute operations within their granted permission levels. While the current implementation relies on database-level permissions, the architecture supports query-level filtering.

Permission Levels

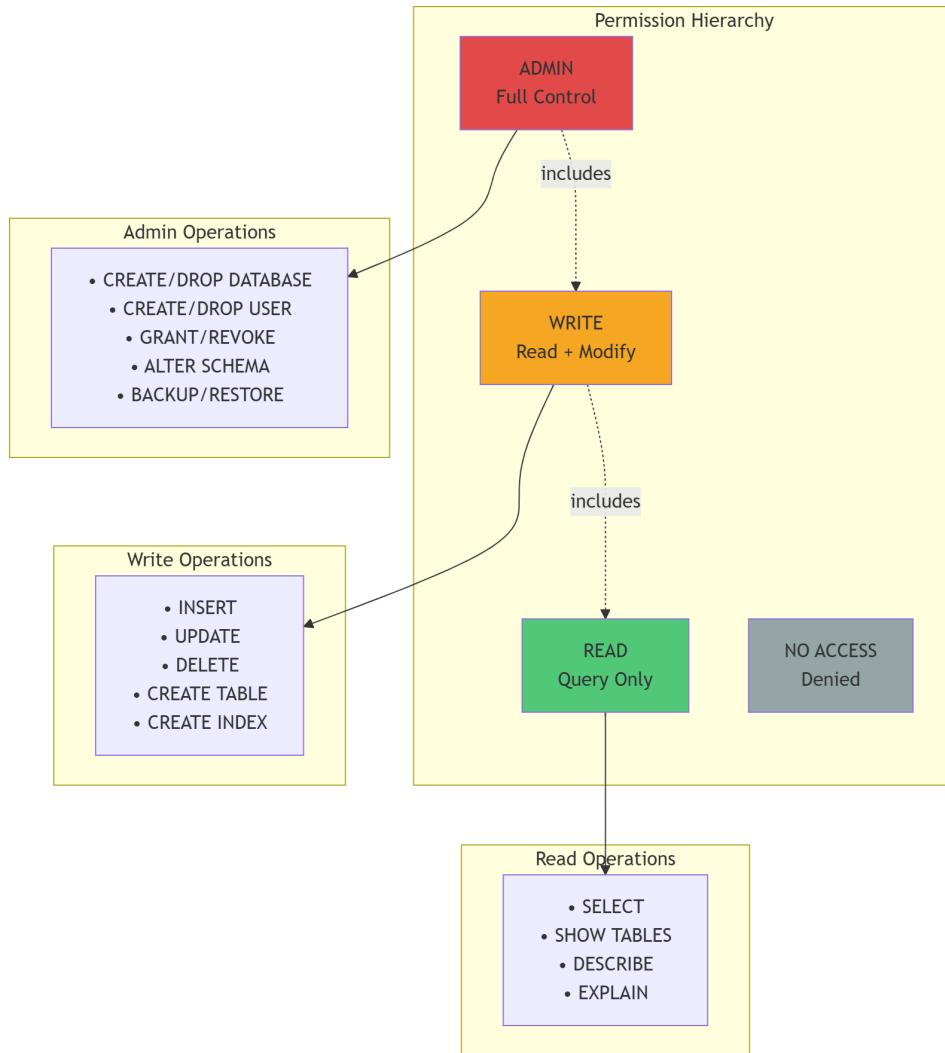


Figure 11.6: Permission levels: read, write, admin

Permission Enforcement Flow

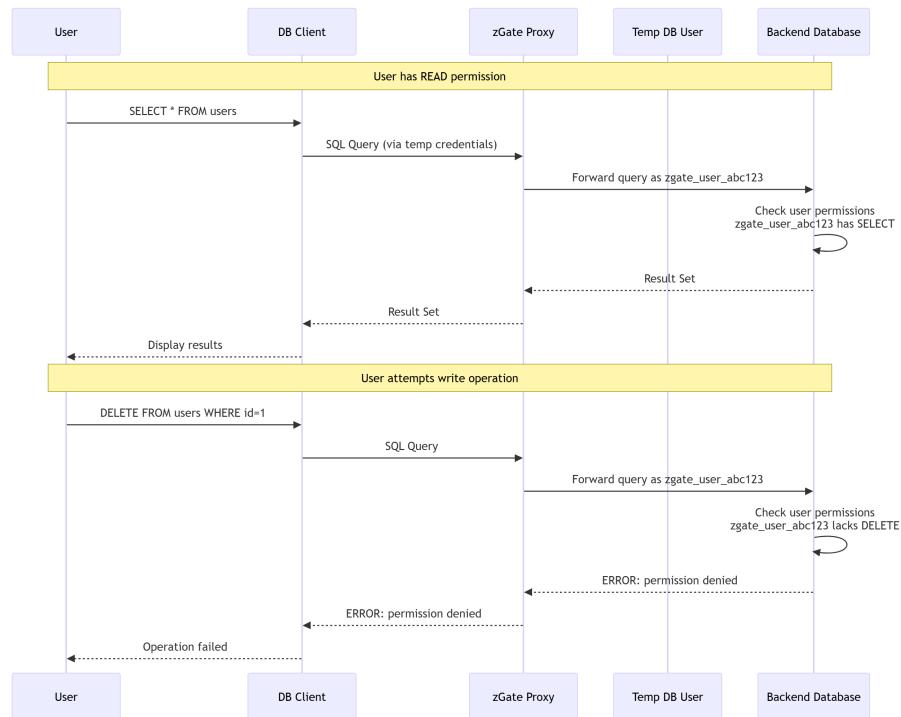


Figure 11.7: Permission enforcement flow

Permission Mapping

The proxy maps zGate permission levels to database-specific grants:

MSSQL Permission Mapping

zGate Level	MSSQL Grants
read	SELECT, VIEW DEFINITION
write	SELECT, INSERT, UPDATE, DELETE
admin	CONTROL, ALTER, CREATE TABLE, DROP TABLE

Table 11.1: MSSQL permission mapping

MySQL Permission Mapping

zGate Level	MySQL Grants
read	SELECT, SHOW VIEW
write	SELECT, INSERT, UPDATE, DELETE
admin	ALL PRIVILEGES

Table 11.2: MySQL permission mapping

11.3.1 Temporary User Creation

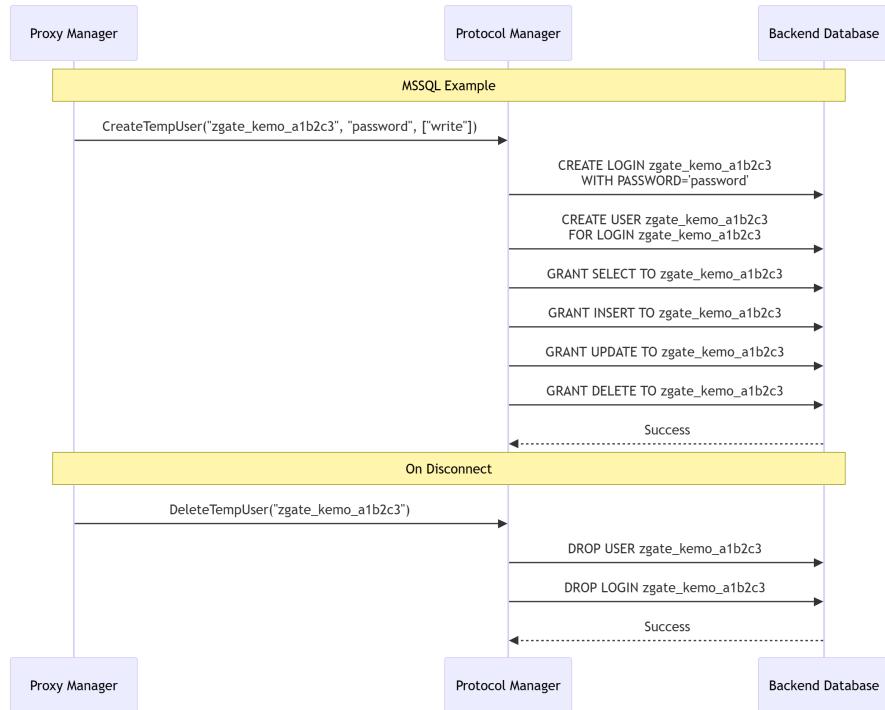


Figure 11.8: Temporary user creation flow

11.4 Policy Enforcement Flow

Policy enforcement is the critical security layer that determines whether a user can access a specific database based on real-time configuration evaluation.

11.4.1 Policy Architecture

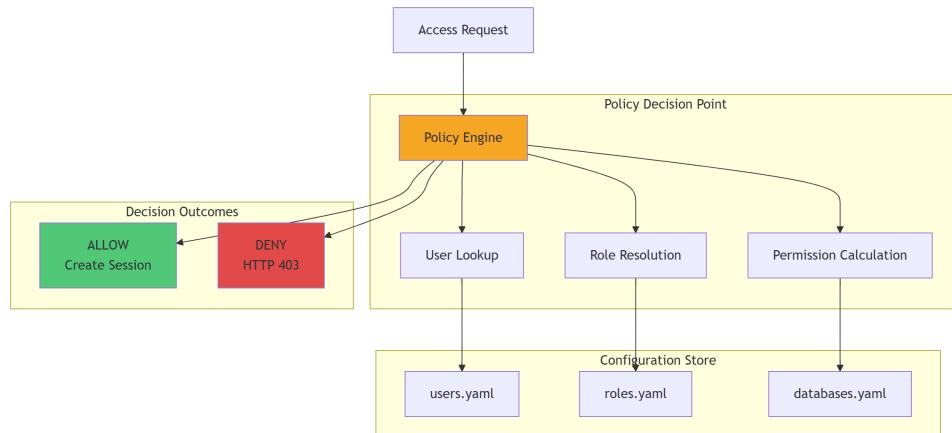


Figure 11.9: Policy enforcement architecture

11.4.2 Real-Time Policy Evaluation

The policy engine always evaluates permissions using the **current** configuration, not the permissions embedded in the JWT token. This ensures immediate effect when permissions are changed.

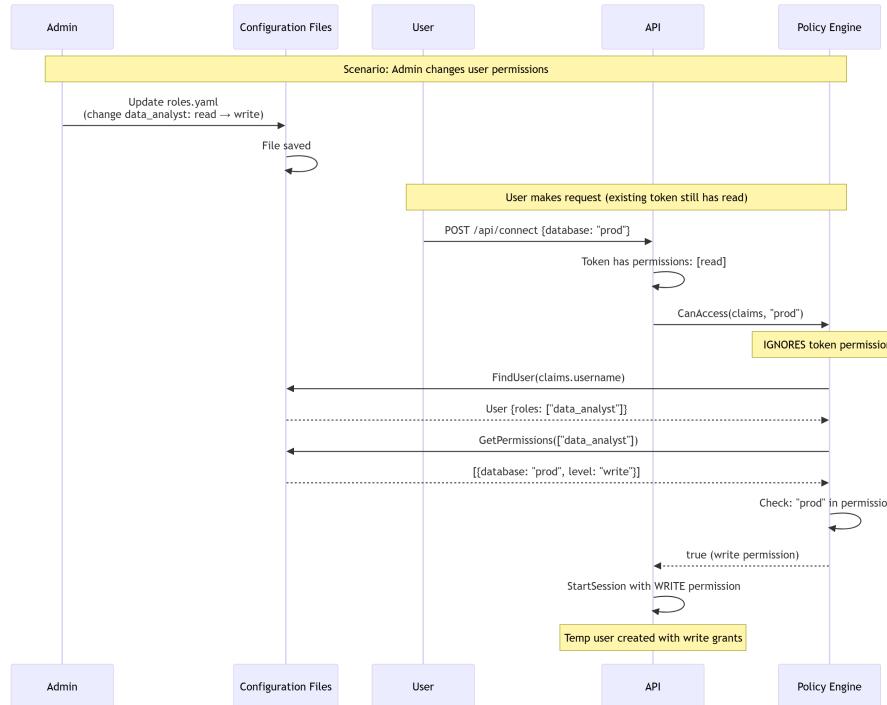


Figure 11.10: Real-time policy evaluation

11.4.3 Policy Enforcement Points

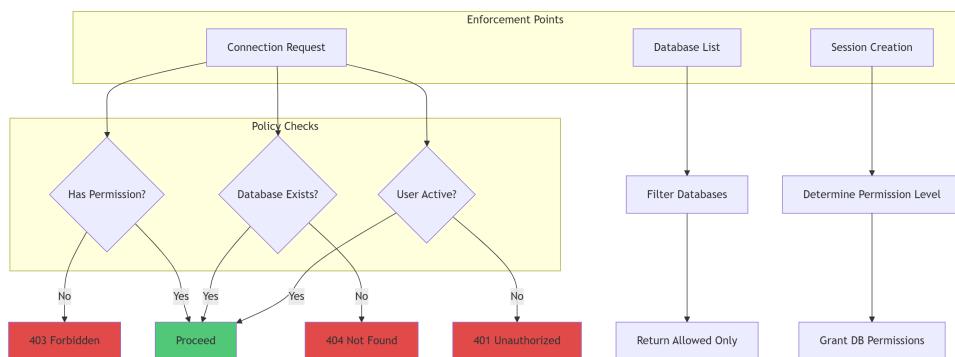


Figure 11.11: Policy enforcement points in the connection lifecycle

11.4.4 End-to-End Flow Narrative

Phase 1: Authentication & Authorization

1. User executes `zgate login` via CLI

2. **CLI** sends credentials to **LoginAPI**
3. **Authenticator** validates against `users.yaml`
4. Password verified using bcrypt comparison
5. **Authenticator** resolves permissions from `roles.yaml`
6. JWT tokens generated and returned to **CLI**
7. **CLI** stores tokens in OS keyring

Phase 2: Database Discovery

1. **User** executes `zgate list` via **CLI**
2. **CLI** sends request with token to **ListAPI**
3. **AuthMiddleware** validates JWT token
4. **PolicyEngine** looks up fresh permissions from `users.yaml + roles.yaml`
5. **PolicyEngine** filters `databases.yaml` based on permissions
6. Accessible databases returned to **CLI**
7. **CLI** displays formatted list to **User**

Phase 3: Connection Establishment

1. **User** executes `zgate connect --database prod_db` via **CLI**
2. **CLI** sends request to **ConnectAPI** with token
3. **AuthMiddleware** validates token, extracts claims
4. **PolicyEngine** performs real-time permission check
5. **ConnectAPI** calls **ProxyManager.StartSession()**
6. **ProxyManager** allocates dynamic port (e.g., 5001)
7. **ProtocolFactory** creates appropriate manager (MSSQL/MySQL)
8. **MSSQLMgr/MySQLMgr** generates temp credentials
9. **Manager** creates temporary user in **Backend Database**
10. **Manager** grants permissions based on user's level

11. **ProxyManager** creates **Session** object
12. **Session** starts **DynamicListener** on allocated port
13. Connection details returned to **CLI**
14. **CLI** displays connection instructions to **User**

Phase 4: Active Database Usage

1. **User** uses **DBClient** (e.g., mysql -h localhost -P 5001 -u zgate_kemo_abc123)
2. **DBClient** connects to **DynamicListener** on port 5001
3. **DynamicListener** accepts connection via **ConnectionAcceptor**
4. **ProtocolHandler** establishes backend connection to **MSSQL/MySQL**
5. Database traffic proxied bidirectionally
6. All queries executed as **TempUser** with restricted permissions
7. **Backend Database** enforces grants (read/write/admin)
8. Results returned through proxy to **DBClient**

Phase 5: Session Termination

1. **User** exits **DBClient** or executes `zgate disconnect`
2. **CLI** sends disconnect request to **DisconnectAPI**
3. **ProxyManager.StopSession()** called
4. **Session** context cancelled
5. **DynamicListener** stops accepting connections
6. **Manager** drops temporary user from **Backend Database**
7. **Manager** closes database connection
8. Port released for reuse
9. **Session** removed from **ProxyManager** map
10. Success response returned to **CLI**

12. High-Level Data Flow Diagrams

12.1 Authentication Flow Diagram

12.2 Query Filtering Diagram

12.3 Logging & Auditing Flow Diagram

13. Technology Justification

13.1 Why Go

13.2 Why Node.js / TS / React

13.3 Why mTLS (and why TCP is temporary)

13.4 Why SQLite / Internal Storage

13.5 Design Decision Summary

14. Prototype – Semester 1

14.1 Implemented Features

14.2 Screenshots (CLI & Dashboard)

14.3 What Works vs What Doesn't

14.4 Technical Decisions Made

14.5 Implementation Challenges

15. Development Methodology

15.1 Agile Scrum Framework

To manage the complexity of the project and ensure continuous development, the team adopted the Agile Scrum methodology. We structured the development lifecycle into one-week sprints, enabling rapid iteration and frequent feedback cycles. This short sprint duration allowed us to demonstrate tangible progress weekly and incorporate supervisor feedback more frequently, ensuring alignment with project objectives throughout the development process.

15.2 Meeting Structure

Our workflow is organized around three distinct meeting types: the Weekly Kick-off, Daily Stand-ups, and the Sprint Review with stakeholders. This structured rhythm of recurring meetings ensures that team milestones and progress are consistently monitored and synchronized. Collectively, these meetings serve to define, review, and align all weekly tasks in accordance with agile best practices.

15.2.1 Weekly Kick-off Meeting

This meeting is held at the start of every sprint to align the team for the upcoming week. It encompasses three key components:

- **Retrospective:** We briefly analyze the previous sprint, discussing what went well and identifying bottlenecks (e.g., merge conflicts or unclear requirements). This reflection helps the team avoid repeating past mistakes and continuously improve our process.
- **Backlog Refinement:** We review upcoming tasks to ensure they are clearly defined and that all technical requirements are understood before assignment. This step reduces ambiguity and sets clear expectations.
- **Sprint Planning:** We select specific tasks from the refined backlog to be completed in the current week. Tasks are assigned to team members based on priority,

complexity, and estimated effort.

15.2.2 Daily Stand-up Meeting

This brief synchronization meeting is held daily to maintain continuous team alignment and identify blockers early.

- **Duration:** Limited to approximately 15 minutes total, with each member speaking for no more than 2 minutes. This time constraint ensures the meeting remains focused and efficient.
- **Format:** Each team member addresses two specific points: what they accomplished yesterday and what they plan to work on today. Any blockers or dependencies are also raised.
- **Objective:** This practice ensures that no team member works in isolation or is blocked by dependencies without the rest of the team being aware. It promotes transparency and enables rapid problem-solving.

15.2.3 Sprint Review (Weekly Supervisor Meeting)

At the conclusion of each sprint, a formal review is conducted with our project supervisor and mentors to validate progress and gather feedback.

- **Demonstration:** The team presents the functional features completed during the sprint, showcasing working software rather than just documentation or plans.
- **Validation:** Our supervisors provide immediate feedback on the implementation. This feedback is either approved for integration or converted into new change requests to be prioritized in the next sprint's backlog.

15.3 Collaboration Tools

To ensure accessibility and efficient collaboration across all phases of development, we employ an integrated tool stack that supports both synchronous and asynchronous communication:

Central Knowledge Base (Notion): Serves as the single source of truth for the project wiki. It is used for sprint planning, task and goal tracking, documenting new code features, and archiving meeting notes and recordings. All team members have real-time access to project documentation.

Version Control & Technical Documentation (GitHub): The primary repository for source code, version control, and code reviews. GitHub also hosts technical documentation, including the Proxy Installation Guide and API references.

Synchronous Communication: Discord, Microsoft Teams, and Google Meet are used for scheduled meetings and real-time voice/video collaboration. These platforms facilitate immediate discussion and decision-making.

Asynchronous Communication: WhatsApp is used for quick, urgent team notifications and simple coordination when immediate responses are needed outside of scheduled meetings.

Auxiliary Tools: We leverage AI-powered tools for generating meeting summaries and conclusions, creating immediate and searchable records of discussions. Excalidraw is used for creating collaborative diagrams, flowcharts, and architectural drawings during design discussions.

15.4 Documentation & Observability

We prioritize high observability by documenting all progress, regardless of scale. This comprehensive documentation approach ensures transparency and facilitates knowledge transfer within the team.

While day-to-day execution is tracked in Notion, high-level progress reporting is documented in a formal **Supervisor Meeting Log** maintained as a Word document. This log serves as an official record and tracks:

- **Attendance & Date:** A record of meeting participants and timestamps.
- **Retrospective:** Feedback on completed tasks, including what was delivered and any deviations from the plan.
- **Forward Planning:** Clearly defined goals and expected outcomes for the subsequent meeting, establishing accountability and measurable targets.

16. Task Tracking

16.1 Team Task Tracking (Actual Examples)

16.2 Supervisor Tracking Logs

16.3 Blockers, Risks & Resolution Notes

17. Milestones

17.1 Term 1 Milestone Roadmap

17.1.1 Milestone 1

17.1.2 Milestone 2

17.1.3 Milestone 3

17.1.4 Milestone 4

17.1.5 Milestone 5

17.2 Timeline Chart (Gantt-like)

18. Threat Model & Security Considerations

18.1 Threat Model

18.2 Risks & Attack Vectors

18.3 Mitigation Techniques

18.4 Why Zero Trust is Needed

19. Roadmap for Term 2

19.1 Remaining Features

19.2 Architecture Improvements

19.3 Performance Goals

19.4 Testing & Validation Plan

20. Team Contribution

20.1 Overview of Contribution Approach

20.2 Individual Contributions

21. Expected Outcomes

22. Conclusion

- 22.1 Restated Purpose**
- 22.2 Summary of Achievements**
- 22.3 Importance & Contribution**
- 22.4 Transition to Next Semester**

References

Bibliography

A. Glossary

B. Dashboard Mockups