TeamSphere Enterprise HR Management System

Complete Technical Documentation

Version: 1.0.0

Date: October, 2025

Classification: Confidential - Internal Use

Prepared By: TeamSphere Development Team

Reviewed By: Technical Architecture Board

Approved By: Chief Technology Officer

I. Executive Summary

1.1 System Overview

TeamSphere is an enterprise-grade, multi-tenant Human Resource Management System (HRMS) designed to streamline workforce management for modern businesses. The platform provides comprehensive time tracking, employee management, leave management, and compliance features across web and mobile applications.

Attribute	Description
Platform Type	SaaS / On-Premise HRMS
Target Market	Small to Enterprise-sized businesses (20-Unlimited employees)
Deployment Models	Cloud SaaS, On-Premise Docker, On-Premise Kubernetes
Technology Stack	Java Spring Boot backend, React frontend, React Native mobile apps
Primary Database	PostgreSQL 14+
Architecture Pattern	Multi-tenant with Row-Level Isolation

1.2 Business Objectives

TeamSphere addresses critical challenges in workforce management by providing a comprehensive solution that ensures accuracy, compliance, and operational efficiency. The platform is designed to support organizations of varying sizes with flexible deployment options tailored to their specific requirements.

Primary Objectives:

- Provide accurate, real-time employee time tracking
- Ensure regulatory compliance (SOX, GDPR, ISO 27001)
- Reduce administrative burden through automation
- Eliminate buddy punching and time theft through device binding
- Enable data-driven workforce decisions
- Support flexible deployment models (cloud and on-premise)

Target Customer Segments:

Segment	Recommended Solution
SMBs (20-100 employees)	Cloud SaaS subscription model with monthly pricing ($49-$149/month)
Mid-Market (100-500 employees)	On-premise deployment or dedicated cloud instances ($149-$299/month)
Enterprise (500+ employees)	On-premise Kubernetes with high availability and custom SLAs ($299-$399/month)

Revenue Model:

Revenue Stream	Pricing	Description
SaaS Subscriptions	$49-$399/month	5 pricing tiers based on employee count and features
On-Premise Licenses	$490-$3,990	One-time perpetual license with optional support contracts
Support Contracts	30% of license fee annually	Includes updates, patches, and technical support
Professional Services	Custom pricing	Installation, customization, integration, and training

1.3 Key Features & Differentiators

TeamSphere distinguishes itself in the HRMS marketplace through innovative architectural decisions and features that directly address common pain points in workforce management. The following differentiators provide significant competitive advantages:

1. Immutable Attendance Architecture

Unlike traditional HR systems that allow modification or deletion of time records, TeamSphere implements an immutable data architecture where original clock-in/clock-out records are never altered. Any HR corrections are stored as separate audit records with complete chain of custody. This approach ensures:

- Full compliance with SOX Section 404 requirements for internal controls
- GDPR Article 5 compliance for data accuracy and integrity
- ISO 27001 alignment for information security management
- Complete audit trail for regulatory inspections and legal disputes
- Prevention of time theft and fraudulent record manipulation

2. Three-Deployment Model Flexibility

TeamSphere uniquely supports three distinct deployment architectures from a single codebase, allowing organizations to choose the model that best fits their requirements:

Model	Architecture	Key Benefits	Ideal For
SaaS Cloud	Multi-tenant shared infrastructure	Instant deployment, automatic updates, lowest TCO	SMBs, rapid deployment needs
On-Premise Docker	Single-tenant containerized deployment	Data sovereignty, custom networking, air-gapped environments	Mid-market, regulatory requirements
On-Premise Kubernetes	Enterprise-grade orchestrated deployment	High availability, auto-scaling, disaster recovery	Large enterprises, mission-critical systems

3. 04:00 AM Workday Boundary

TeamSphere introduces an industry-first workday calculation algorithm that correctly handles overnight shifts spanning two calendar days. The 04:00 AM workday boundary ensures that:

- Employees clocking in at 11:00 PM on December 31 and out at 7:00 AM January 1 have hours attributed to December 31
- Manufacturing, healthcare, security, and hospitality industries accurately track overnight operations
- Payroll calculations align with actual work performed rather than arbitrary calendar boundaries
- Compliance with labor regulations requiring accurate shift attribution

4. Device Fingerprinting & Anti-Fraud Technology

TeamSphere implements advanced device fingerprinting using 15+ parameters to create unique device signatures. This technology prevents buddy punching and time theft through:

- Browser fingerprinting (user agent, screen resolution, timezone, language, plugins)
- Hardware fingerprinting (GPU renderer, CPU cores, memory, touch support)
- Network fingerprinting (IP address, geolocation with consent)
- Device binding requiring initial registration and approval
- Anomaly detection for suspicious patterns (e.g., same employee clocking from multiple cities)
- GDPR-compliant consent management for location tracking

1.4 Technology Stack Summary

TeamSphere is built on modern, enterprise-grade technologies selected for their reliability, scalability, and extensive ecosystem support. The technology stack follows industry best practices and utilizes Long-Term Support (LTS) versions where applicable.

Backend Technology Stack

Component	Technology & Version
Framework	Spring Boot 3.x
Language	Java 17 LTS
Database	PostgreSQL 14+
ORM	JPA/Hibernate 6.x
Security	Spring Security + JWT 6.x
Migration	Flyway 9.x
API Docs	Swagger/OpenAPI 3.0
Build Tool	Maven 3.8+

Frontend Technology Stack

Component	Technology & Version
Framework	React 18.x
Build Tool	Vite 4.x
UI Library	Material-UI (MUI) 5.x
Routing	React Router 6.x
State Management	React Context + Query 4.x
HTTP Client	Axios 1.x
Internationalization	i18next 23.x

Mobile Technology Stack

Component	Technology & Version
Framework	React Native 0.73
Language	TypeScript 5.x
Navigation	React Navigation 6.x
NFC	react-native-nfc-manager
Storage	AsyncStorage 1.x
Camera	react-native-camera

Infrastructure & DevOps Stack

Component	Technology/Purpose
Containerization	Docker
Orchestration	Kubernetes
Database	PostgreSQL
Cache	Redis (planned)
Monitoring	Prometheus + Grafana
Logging	ELK Stack (planned)
CI/CD	GitHub Actions
IaC	Terraform (planned)

The technology stack is designed for long-term maintainability with active community support, extensive documentation, and proven enterprise adoption. All major components use semantic versioning and follow predictable release cycles.

II. System Architecture

2.1 High-Level Architecture

TeamSphere implements a modern three-tier architecture pattern separating presentation, application logic, and data persistence layers. This architectural approach provides clear separation of concerns, facilitates independent scaling of components, and enables parallel development across teams.

Architectural Layers:

Layer	Technology	Responsibilities
Presentation Layer	Web (React + Vite), Mobile iOS/Android (React Native)	User interface, client-side validation, state management
Application Layer	Spring Boot REST API (Java 17)	Business logic, authentication, authorization, multi-tenancy
Data Layer	PostgreSQL 14+	Data persistence, ACID transactions, row-level security
Integration Layer	REST APIs, External Services	Payment processing, email, cloud storage, monitoring

<Insert System Architecture Diagram Here>

Figure 1: High-level system architecture showing the interaction between presentation, application, and data layers with external integrations.

Architectural Patterns & Principles:

- Layered Architecture: Clear separation between presentation, business, and data access layers
- Repository Pattern: Abstraction of data access logic with Spring Data JPA repositories
- Service Layer Pattern: Business logic encapsulated in service classes with transaction management
- Dependency Injection: Spring Framework IoC container manages component lifecycle and dependencies
- RESTful API Design: Stateless HTTP-based APIs following REST architectural constraints
- Multi-Tenancy: Row-level data isolation using tenant discriminator (company_id)
- Immutability Pattern: Time-sensitive data stored in append-only immutable tables

2.2 Backend Architecture (Spring Boot)

The backend is built using Spring Boot 3.x framework, leveraging the Spring ecosystem for enterprise-grade features including dependency injection, transaction management, security, and data access. The application follows a layered architecture with clear boundaries between components.

Backend Layer Structure:

Layer	Annotation	Responsibilities
Controller Layer	@RestController	HTTP request handling, request validation, response formatting
Service Layer	@Service	Business logic, transaction management, service orchestration
Repository Layer	@Repository	Data access, CRUD operations, custom queries
Entity Layer	@Entity	JPA entities, object-relational mapping, validation
Security Layer	Spring Security	Authentication, authorization, JWT handling
Configuration Layer	@Configuration	Bean definitions, application configuration

Spring Boot Features Utilized:

- Spring Boot Starter Web: RESTful API development with embedded Tomcat server
- Spring Data JPA: Repository pattern implementation with Hibernate ORM
- Spring Security: Authentication, authorization, and CSRF protection
- Spring Boot Actuator: Health checks, metrics, and application monitoring endpoints
- Spring Validation: Bean validation using JSR-303 annotations
- Spring Transaction Management: Declarative transaction handling with @Transactional
- Spring AOP: Cross-cutting concerns (logging, auditing) using aspect-oriented programming
- Flyway Migration: Database schema versioning and migration management

Multi-Tenancy Implementation:

TeamSphere implements row-level multi-tenancy using a tenant discriminator column (company_id) present in all tenant-aware tables. The TenantContext class maintains the current tenant ID in a ThreadLocal variable, ensuring thread-safe tenant isolation.

Sample Multi-Tenancy Code:

```
public class TenantContext {
    private static ThreadLocal<Long> currentCompanyId = new ThreadLocal<>();

    public static void setCurrentCompanyId(Long companyId) {
        currentCompanyId.set(companyId);
    }

    public static Long getCurrentCompanyId() {
        return currentCompanyId.get();
    }

    public static void clear() {
        currentCompanyId.remove();
    }
}

@Component
public class TenantFilter extends OncePerRequestFilter {
    @Override
    protected void doFilterInternal(HttpServletRequest request,
                                    HttpServletResponse response,
                                    FilterChain filterChain) throws ServletException,
IOException {
        String token = jwtUtil.extractToken(request);
        Long companyId = jwtUtil.getCompanyId(token);
        TenantContext.setCurrentCompanyId(companyId);

        try {
            filterChain.doFilter(request, response);
        } finally {
            TenantContext.clear();
        }
    }
}
```

2.3 Frontend Architecture (React)

The web frontend is built using React 18.x with Vite as the build tool, providing fast development experience and optimized production builds. The application follows a component-based architecture with clear separation between presentational and container components.

Layer	Description	Technology
Component Layer	Reusable UI components (buttons, forms, tables, dialogs)	Material-UI (MUI) components
Page/View Layer	Complete application pages (Dashboard, Employees, Time Logs)	React Router for navigation
State Management	Global and local state handling	React Context + React Query
API Layer	HTTP client for backend communication	Axios with interceptors
Routing Layer	Client-side routing and navigation	React Router v6
Internationalization	Multi-language support	i18next library

State Management Strategy:

State Type	Solution	Use Cases
Server State	React Query	API data, caching, automatic refetching
Global App State	React Context	Authentication, user preferences, theme, language
Component State	useState Hook	Local UI state, form inputs, modal visibility
Form State	React Hook Form	Form validation, error handling, submission
URL State	React Router	Navigation parameters, query strings

Key Frontend Features:

- Responsive design supporting desktop, tablet, and mobile browsers
- Material Design UI components for consistent look and feel
- Real-time data updates using React Query automatic refetching
- Form validation with user-friendly error messages
- Multi-language support (English, Turkish, German, French)
- Dark mode and light mode theme switching
- Progressive Web App (PWA) capabilities for offline access
- Accessibility (WCAG 2.1 Level AA compliance)

2.4 Mobile Architecture (React Native)

The mobile applications for iOS and Android are built using React Native 0.73, enabling code sharing between platforms while maintaining native performance. The architecture leverages platform-specific features including NFC, GPS, and camera through native modules.

Component	Description	Technology
Screen Components	Complete mobile screens (Login, Clock In/Out, Leave Requests)	React Navigation
Shared Components	Reusable UI elements (buttons, inputs, cards)	Custom components
Services Layer	NFC, GPS, Camera, Device Fingerprinting	Native modules
API Integration	Backend communication with token management	Axios
Local Storage	Offline data persistence	AsyncStorage
Navigation	Screen routing and deep linking	React Navigation v6

Native Features Integration:

- NFC Tag Reading: Employee badge scanning for clock in/out operations
- GPS Location Services: Geolocation capture with user consent (GDPR compliant)
- Camera Access: Photo capture for attendance verification and document upload
- Device Fingerprinting: Unique device identification using hardware/software parameters
- Push Notifications: Real-time alerts for leave approvals, shift reminders
- Biometric Authentication: Face ID / Touch ID for secure app access
- Background Location: Geofencing for automatic clock in/out (with consent)

2.5 Data Flow & Integration Layers

The system implements a request-response data flow pattern with clear boundaries between client applications and the backend API. All client-server communication uses RESTful HTTP(S) with JSON payloads and JWT-based authentication.

Complete Request Flow (Clock-In Example):

```
1. User Action: Employee taps 'Clock In' button in mobile app

2. NFC Scan: Read employee badge NFC tag ID

3. GPS Location: Capture current coordinates (with user consent)

4. Device Fingerprint: Calculate unique device signature

5. API Request:
   POST /api/time-logs/clock-in
   Headers: { Authorization: "Bearer <JWT_TOKEN>" }
   Body: {
     "employeeId": 123,
     "nfcTagId": "E4:3C:2A:...",
     "location": { "lat": 40.7128, "lon": -74.0060 },
     "deviceFingerprint": "a3f5c8d9..."
   }

6. Backend Processing:
   a. JWT Filter → Extract companyId, set TenantContext
   b. Controller → Validate request (@Valid annotation)
   c. Service → Verify employee exists and is active
   d. Service → Check device binding (prevent buddy punching)
   e. Service → Verify NFC tag matches employee
   f. Service → Create TimeLog entity
   g. Repository → Save to database
   h. Audit Service → Log action for compliance

7. Database Write:
   INSERT INTO time_logs (
     company_id, employee_id, clock_in,
     location_lat, location_lon, device_fingerprint
   ) VALUES (1, 123, '2025-10-23 08:00:00', ...)

8. API Response:
   Status: 200 OK
   Body: {
     "success": true,
     "message": "Clocked in successfully",
     "data": {
       "timeLogId": 456,
       "clockIn": "2025-10-23T08:00:00Z"
     }
   }

9. Mobile UI Update:
   - Display success notification
   - Update status to "Clocked In"
   - Start duration timer
```

External Integrations:

Service	Purpose	Protocol	Use Cases
Stripe	Payment processing for SaaS subscriptions	REST API	Monthly billing, invoicing
SMTP Server	Email notifications	SMTP Protocol	Leave approvals, password reset
AWS S3 / Azure Blob	File storage for uploads	REST API	Profile photos, documents
Prometheus	Metrics collection	HTTP Pull	Application metrics, alerts
Grafana	Monitoring dashboards	Prometheus Data Source	System health visualization
Sentry (Planned)	Error tracking	SDK Integration	Exception monitoring

2.6 Multi-Tenancy Architecture

TeamSphere implements row-level multi-tenancy (also known as shared database, shared schema) where all tenants share the same database tables, but data is logically isolated using a tenant discriminator column (company_id). This approach provides optimal resource utilization while maintaining complete data isolation.

Multi-Tenancy Design Decisions:

Aspect	Implementation	Rationale
Isolation Level	Row-Level (Shared Schema)	Balances resource efficiency with data isolation
Discriminator	company_id BIGINT	Every tenant-aware table includes this foreign key
Enforcement	Application + Database	Spring Data JPA filters + PostgreSQL Row-Level
Tenant Context	ThreadLocal variable	Maintains current tenant ID per request thread
Security	JWT token includes company_id	Client cannot access other tenant data
Performance	Indexed company_id columns	Fast query filtering with proper indexes

Database-Level Protection:

```
-- Row-Level Security (Optional layer of protection)
ALTER TABLE employees ENABLE ROW LEVEL SECURITY;

CREATE POLICY tenant_isolation ON employees
USING (company_id = current_setting('app.current_company_id')::bigint);

-- Ensures all queries automatically filter by tenant
-- Even if application code has bugs, database enforces isolation
```

Application-Level Enforcement:

```
// Base repository with automatic tenant filtering
public interface TenantAwareRepository<T extends BaseEntity, ID>
    extends JpaRepository<T, ID> {

    default List<T> findAll() {
        return findByCompanyId(TenantContext.getCurrentCompanyId());
    }

    List<T> findByCompanyId(Long companyId);
}

// Entity base class
@MappedSuperclass
public abstract class BaseEntity {
    @Column(name = "company_id", nullable = false, updatable = false)
    private Long companyId;

    @PrePersist
    protected void onPersist() {
        if (companyId == null) {
            companyId = TenantContext.getCurrentCompanyId();
        }
    }
}
```

III. Deployment & Environment Setup

3.1 Deployment Models Overview

TeamSphere supports three distinct deployment models, each designed for specific organizational requirements, compliance needs, and technical capabilities. The application is packaged as container images, enabling consistent deployment across all environments.

Model	Tenancy	Infrastructure	Resources	Management	Setup Time	Target
SaaS Cloud	Multi-tenant	AWS/ Azure/GCP	Shared	Managed by TeamSphere	Minutes	SMBs, startups
On-Premise Docker	Single-tenant	Customer datacenter	Dedicated	Customer managed	1-2 hours	Mid-market, compliance
On-Premise K8s	Single-tenant	Customer datacenter	Dedicated + HA	Customer managed	4-8 hours	Enterprise, mission-critical

3.2 SaaS Cloud Deployment

The SaaS deployment model provides a fully managed, multi-tenant environment hosted on cloud infrastructure. This model offers the fastest time-to-value with zero infrastructure management overhead for customers.

SaaS Architecture Components:

Component	Technology	Purpose
Load Balancer	AWS ALB / Azure App Gateway	SSL termination, traffic distribution
Application Servers	ECS Fargate / Azure Container Instances	Auto-scaling Spring Boot containers (2-20 instances)
Database	Amazon RDS PostgreSQL / Azure DB for PostgreSQL	Multi-AZ deployment, automated backups
Cache Layer	Amazon ElastiCache / Azure Cache for Redis	Session storage, application cache
Object Storage	Amazon S3 / Azure Blob Storage	File uploads, backups, static assets
Monitoring	CloudWatch / Azure Monitor	Metrics, logs, alerts
CDN	CloudFront / Azure CDN	Static asset delivery, global caching

SaaS Features:

- Automatic scaling based on demand (2-20 application instances)
- 99.9% uptime SLA with multi-AZ deployment
- Automated daily backups with 30-day retention
- Automated security patching and updates
- Built-in DDoS protection and WAF
- Global CDN for fast asset delivery
- Continuous monitoring and alerting
- Disaster recovery with cross-region replication

3.3 On-Premise Docker Deployment

The Docker deployment model provides a containerized single-tenant installation suitable for organizations with on-premise infrastructure requirements, air-gapped environments, or specific data sovereignty regulations.

Docker Compose Configuration:

```
version: '3.8'

services:
  teamsphere-backend:
    image: teamsphere/backend:1.0.0
    container_name: teamsphere-api
    ports:
      - "8080:8080"
    environment:
      - SPRING_DATASOURCE_URL=jdbc:postgresql://postgres:5432/teamsphere
      - SPRING_DATASOURCE_USERNAME=teamsphere_user
      - SPRING_DATASOURCE_PASSWORD=${DB_PASSWORD}
      - JWT_SECRET=${JWT_SECRET}
      - COMPANY_ID=${COMPANY_ID}
    depends_on:
      - postgres
    restart: unless-stopped

  teamsphere-frontend:
    image: teamsphere/frontend:1.0.0
    container_name: teamsphere-web
    ports:
      - "80:80"
      - "443:443"
    environment:
      - VITE_API_URL=http://teamsphere-api:8080
    depends_on:
      - teamsphere-backend
    restart: unless-stopped

  postgres:
    image: postgres:14
    container_name: teamsphere-db
    ports:
      - "5432:5432"
    environment:
      - POSTGRES_DB=teamsphere
      - POSTGRES_USER=teamsphere_user
      - POSTGRES_PASSWORD=${DB_PASSWORD}
    volumes:
      - postgres_data:/var/lib/postgresql/data
      - ./backups:/backups
    restart: unless-stopped

volumes:
  postgres_data:
```

Installation Steps:

1. 1. Install Docker Engine (20.10+) and Docker Compose (v2.0+)
2. 2. Create .env file with required environment variables (DB_PASSWORD, JWT_SECRET, COMPANY_ID)
3. 3. Pull TeamSphere container images from registry
4. 4. Run 'docker-compose up -d' to start all services
5. 5. Access application at https://localhost after initialization
6. 6. Configure SSL certificates (Let's Encrypt or corporate CA)
7. 7. Set up automated backup cron jobs
8. 8. Configure firewall rules and network security

3.4 On-Premise Kubernetes Deployment

The Kubernetes deployment model provides enterprise-grade orchestration with high availability, auto-scaling, rolling updates, and self-healing capabilities. This model is recommended for organizations with mission-critical requirements and existing Kubernetes infrastructure.

Kubernetes Architecture:

Resource	Configuration	Purpose
Deployment (Backend)	3+ replicas	Spring Boot API pods with auto-scaling
Deployment (Frontend)	2+ replicas	Nginx serving React application
StatefulSet (Database)	3-node cluster	PostgreSQL with streaming replication
Service (LoadBalancer)	External IP	Ingress controller (nginx/ traefik)
ConfigMap	Configuration	Non-sensitive environment variables
Secret	Sensitive data	Database passwords, JWT secrets
PersistentVolumeClaim	Storage	Database data, file uploads
HorizontalPodAutoscaler	Auto-scaling	CPU/memory-based scaling (2-10 pods)

Sample Kubernetes Deployment:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: teamsphere-backend
  namespace: teamsphere
spec:
  replicas: 3
  selector:
    matchLabels:
      app: teamsphere-backend
  template:
    metadata:
      labels:
        app: teamsphere-backend
    spec:
      containers:
      - name: backend
        image: teamsphere/backend:1.0.0
        ports:
        - containerPort: 8080
        env:
        - name: SPRING_DATASOURCE_URL
          valueFrom:
            configMapKeyRef:
              name: teamsphere-config
              key: database_url
        - name:
SPRING_DATASOURCE_PASSWORD
          valueFrom:
            secretKeyRef:
              name: teamsphere-secrets
              key: db_password
        resources:
          requests:
            memory: "512Mi"
            cpu: "250m"
          limits:
            memory: "2Gi"
            cpu: "1000m"
        livenessProbe:
          httpGet:
            path: /actuator/health
            port: 8080
          initialDelaySeconds: 60
          periodSeconds: 10
        readinessProbe:
          httpGet:
            path: /actuator/health/
readiness
            port: 8080
          initialDelaySeconds: 30
          periodSeconds: 5
---
apiVersion: autoscaling/v2
kind: HorizontalPodAutoscaler
metadata:
  name: teamsphere-backend-hpa
  namespace: teamsphere
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: teamsphere-backend
  minReplicas: 3
  maxReplicas: 10
  metrics:
  - type: Resource
    resource:
      name: cpu
      target:
        type: Utilization
        averageUtilization: 70
  - type: Resource
    resource:
      name: memory
      target:
        type: Utilization
        averageUtilization: 80
```

3.5 Environment Variables & Configuration

TeamSphere uses environment variables for configuration management, following the twelve-factor app methodology. This approach separates configuration from code, enabling the same application image to be deployed across multiple environments (development, staging, production).

Variable	Example	Description	Required
SPRING_DATASOURCE_URL	jdbc:postgresql://db:5432/teamsphere	PostgreSQL connection string	Required
SPRING_DATASOURCE_USERNAME	teamsphere_user	Database username	Required
SPRING_DATASOURCE_PASSWORD	<secret>	Database password	Required
JWT_SECRET	<256-bit secret>	JWT signing key	Required
JWT_EXPIRATION_MS	86400000	Token expiration (24 hours)	Optional
COMPANY_ID	1	Default company ID (single-tenant)	On-Premise only
SMTP_HOST	smtp.gmail.com	Email server hostname	Optional
SMTP_PORT	587	Email server port	Optional
S3_BUCKET_NAME	teamsphere-uploads	AWS S3 bucket for uploads	Optional
LOG_LEVEL	INFO	Application log level	Optional

3.6 CI/CD Pipeline

TeamSphere implements automated continuous integration and deployment using GitHub Actions. The pipeline ensures code quality, runs automated tests, builds container images, and deploys to target environments.

CI/CD Workflow Stages:

Stage	Action	Details
1. Code Checkout	Git clone repository	Triggered on push/pull request
2. Build Backend	Maven clean install	Compile Java code, run unit tests
3. Build Frontend	npm run build	Compile React code, optimize bundles
4. Run Tests	JUnit, Jest tests	Backend and frontend test suites
5. Code Quality	SonarQube analysis	Code coverage, security vulnerabilities
6. Build Images	Docker build	Create backend and frontend container images
7. Push Images	Docker push	Upload images to container registry
8. Deploy (Dev)	kubectl apply	Deploy to development environment
9. Integration Tests	Automated E2E tests	Verify deployment success
10. Deploy (Prod)	Manual approval	Production deployment (protected)

IV. Core Functional Modules

4.1 Authentication & Authorization

TeamSphere implements JWT-based authentication with role-based access control (RBAC). The authentication system ensures secure user identity verification while authorization controls access to specific features based on user roles.

Authentication Flow:

```
1. User enters email and password in login form
2. Frontend sends POST /api/auth/login with credentials
3. Backend validates credentials against hashed password in database
4. Backend generates JWT token containing:
   - userId
   - companyId
   - roles (SUPER_ADMIN, ADMIN, HR, MANAGER, EMPLOYEE)
   - expiration (24 hours)
5. Backend returns JWT token + user profile data
6. Frontend stores token in localStorage
7. Subsequent requests include token in Authorization header
8. Backend validates token and extracts user context for each request
```

Role Hierarchy:

Role	Description	Permissions
SUPER_ADMIN	Platform administrator	All permissions across all companies (SaaS only)
ADMIN	Company administrator	Full control within company (user management, settings)
HR	HR manager	Employee management, time logs, leave approvals, reports
MANAGER	Department manager	View team members, approve leaves, view reports
EMPLOYEE	Standard employee	Clock in/out, request leaves, view own data

4.2 Employee Management

The employee management module provides comprehensive employee lifecycle management including onboarding, profile management, department/position assignments, and offboarding. The module supports organizational hierarchy and reporting relationships.

Employee Features:

- Employee profile creation with personal and employment information
- Department and position assignments with reporting hierarchy
- NFC tag binding for clock in/out authentication
- Device registration and fingerprinting for security
- Employment status management (Active, On Leave, Terminated)
- Document uploads (ID, contracts, certificates)
- Organizational chart visualization
- Bulk import from Excel spreadsheets

Employee Data Model:

Field	Data Type	Description
id	BIGINT	Primary key, auto-generated
company_id	BIGINT	Tenant discriminator, foreign key to companies
first_name	VARCHAR(50)	Employee first name
last_name	VARCHAR(50)	Employee last name
email	VARCHAR(100)	Unique email address
phone	VARCHAR(20)	Phone number
department_id	BIGINT	Foreign key to departments
position	VARCHAR(100)	Job title/position
manager_id	BIGINT	Foreign key to employees (self-reference)
hire_date	DATE	Date of hire
nfc_tag_id	VARCHAR(50)	NFC badge identifier
status	ENUM	ACTIVE, ON_LEAVE, TERMINATED
created_at	TIMESTAMP	Record creation timestamp
updated_at	TIMESTAMP	Last update timestamp

4.3 Time Tracking & Attendance

The time tracking module records employee work hours with high accuracy and fraud prevention. The system implements device binding, NFC authentication, and geolocation tracking to ensure authentic clock in/out operations.

Time Tracking Features:

- Clock in/out via web, mobile, or NFC kiosk
- Device fingerprinting to prevent buddy punching
- GPS location capture with configurable geofencing
- 04:00 AM workday boundary for overnight shifts
- Automatic break detection and tracking
- Real-time work duration calculation
- Overtime calculation based on configurable rules
- Late arrival and early departure tracking
- Missing punch detection and alerts

Time Log Data Model:

Field	Data Type	Description
id	BIGINT	Primary key
company_id	BIGINT	Tenant discriminator
employee_id	BIGINT	Foreign key to employees
clock_in	TIMESTAMP	Clock in timestamp
clock_out	TIMESTAMP	Clock out timestamp (nullable)
work_duration_minutes	INTEGER	Calculated work duration
workday_date	DATE	Workday attribution (04:00 AM boundary)
location_lat	DECIMAL(10,8)	GPS latitude
location_lon	DECIMAL(11,8)	GPS longitude
device_fingerprint	VARCHAR(255)	Unique device identifier
nfc_tag_id	VARCHAR(50)	NFC tag used for clock in
is_overtime	BOOLEAN	Overtime flag
created_at	TIMESTAMP	Immutable creation timestamp

4.4 Leave Management

The leave management system handles all types of employee absences including vacation, sick leave, personal time, and unpaid leave. The module supports configurable approval workflows and leave balance tracking.

Leave Management Features:

- Multiple leave types (Annual, Sick, Personal, Unpaid, Maternity/Paternity)
- Annual leave balance calculation based on tenure and policy
- Leave request submission with date range selection
- Multi-level approval workflow (Manager → HR → Admin)
- Email notifications for requests, approvals, and rejections
- Leave calendar view for team availability
- Conflict detection (overlapping requests)
- Leave balance reports and history
- Carry-forward of unused leave (configurable)

Leave Request States:

Status	Description
PENDING	Initial state after submission, awaiting approval
APPROVED_BY_MANAGER	Manager approved, awaiting HR review
APPROVED	Fully approved, leave granted
REJECTED_BY_MANAGER	Manager rejected
REJECTED_BY_HR	HR rejected
CANCELLED	Employee cancelled after approval

V. Additional Features

5.1 Backup & Recovery System

TeamSphere implements automated backup and recovery procedures to protect against data loss. The backup system supports both scheduled automatic backups and on-demand manual backups.

Feature	Schedule/Configuration	Details
Automated Backups	Daily at 02:00 AM	PostgreSQL pg_dump, compressed and encrypted
Retention Policy	30 days	Daily backups for 30 days, monthly for 1 year
Storage Location	S3/Azure Blob	Encrypted storage with versioning enabled
Manual Backups	On-demand	Triggered via admin panel or API
Point-in-Time Recovery	WAL archiving	Restore to any point within retention period
Backup Verification	Weekly	Automated restore test to verify backup integrity

5.2 Excel Import/Export

The Excel integration feature enables bulk employee import and comprehensive data export for reporting and integration with external systems. This feature is critical for initial system migration and regular data analysis.

Import Capabilities:

- Bulk employee creation from Excel spreadsheet
- Template validation (column headers, data types)
- Duplicate detection (email, employee ID)
- Error reporting with row-level feedback
- Preview mode before final import
- Support for .xlsx and .csv formats

Export Capabilities:

- Employee directory export
- Time logs export (date range filtered)
- Leave requests and balances export
- Attendance summary reports
- Payroll data export (hours worked, overtime)
- Custom date range selection

5.3 Wiki Documentation Integration

The integrated wiki system provides a centralised knowledge base for company policies, procedures, and documentation. HR administrators can create and maintain documentation accessible to all employees.

- Rich text editor with formatting support
- Document categories and tags
- Search functionality
- Version history and change tracking
- Access control by role
- Markdown support

VI. Security & Compliance

6.1 Authentication Model

TeamSphere uses JSON Web Tokens (JWT) for stateless authentication. Passwords are hashed using bcrypt with a cost factor of 12, providing strong protection against brute-force attacks.

Security Measures:

- Bcrypt password hashing (cost factor 12)
- JWT token expiration (24 hours default)
- Refresh token mechanism for extended sessions
- Password complexity requirements (min 8 chars, uppercase, lowercase, number, special)
- Account lockout after 5 failed login attempts
- Password reset via secure email token
- Multi-factor authentication (planned)
- Session management and logout

6.2 Authorization & RBAC

Role-Based Access Control (RBAC) ensures users can only access features and data appropriate for their role. Permissions are checked at both the API layer and the UI layer.

Permission	SUPER_ADMIN	ADMIN	HR	MANAGER	EMPLOYEE
View Employees	✓	✓	✓	✓	Own Only
Create Employees	✓	✓	✓	✗	✗
Edit Employees	✓	✓	✓	✗	✗
Delete Employees	✓	✓	✗	✗	✗
View Time Logs	✓	✓	✓	✓	Own Only
Edit Time Logs	✓	✓	✓	✗	✗
Approve Leaves	✓	✓	✓	✓	✗
Manage Settings	✓	✓	✗	✗	✗
View Reports	✓	✓	✓	✓	✗

6.3 Data Encryption

TeamSphere implements encryption for data at rest and in transit to protect sensitive information.

Category	Algorithm	Implementation
Data in Transit	TLS 1.3	All HTTP traffic encrypted with SSL/TLS certificates
Data at Rest	AES-256	Database encryption (PostgreSQL native encryption)
Password Storage	Bcrypt	One-way hashing with salt (cost factor 12)
Sensitive Fields	Application-level	SSN, bank account numbers encrypted before storage
Backup Encryption	AES-256	All backup files encrypted before upload to cloud storage

6.4 Audit Logging

Comprehensive audit logging tracks all critical operations for compliance, security analysis, and troubleshooting. Audit logs are immutable and retained for regulatory compliance.

- User login/logout attempts (successful and failed)
- Employee creation, modification, deletion
- Time log creation and HR corrections
- Leave request submissions and approvals
- Permission changes and role assignments
- Configuration changes
- Data exports and backup operations

6.5 GDPR Compliance

TeamSphere is designed to comply with the General Data Protection Regulation (GDPR) requirements for handling personal data of EU citizens.

GDPR Requirement	TeamSphere Implementation
Right to Access	Employees can download all their personal
Right to Erasure	Data deletion with configurable retention periods
Right to Portability	Export data in machine-readable format (JSON/ CSV)
Consent Management	Explicit consent for GPS tracking and camera usage
Data Minimization	Only collect necessary data for business purposes
Privacy by Design	Data protection built into system architecture
Breach Notification	Automated alerts for security incidents

VII. Monitoring & Maintenance

7.1 Health Check Endpoints

Spring Boot Actuator provides health check endpoints for monitoring system status and dependencies.

Endpoint	Purpose
GET /actuator/health	Overall system health status
GET /actuator/health/liveness	Liveness probe (Kubernetes)
GET /actuator/health/readiness	Readiness probe (Kubernetes)
GET /actuator/health/db	Database connectivity check
GET /actuator/metrics	Application metrics
GET /actuator/info	Application version and build info

7.2 Logging Infrastructure

TeamSphere uses SLF4J with Logback for application logging, supporting multiple log levels and output destinations.

Level	Usage
ERROR	Critical errors requiring immediate attention
WARN	Warning conditions that should be reviewed
INFO	General informational messages (default)
DEBUG	Detailed debugging information
TRACE	Very detailed trace information

VIII. API Reference

TeamSphere exposes RESTful APIs for all major operations. All APIs require JWT authentication via the Authorization header. API documentation is available via Swagger UI at /swagger-ui/.

8.1 Authentication APIs

Endpoint	Description	Response
POST /api/auth/login	User login with email/ password	Returns JWT token + user profile
POST /api/auth/logout	User logout	Invalidates session
POST /api/auth/refresh	Refresh JWT token	Returns new token
POST /api/auth/forgot-password	Request password reset	Sends reset email
POST /api/auth/reset-password	Reset password with token	Updates password

Sample Login Request:

```
POST /api/auth/login
Content-Type: application/json

{
  "email": "john.doe@company.com",
  "password": "SecurePass123!"
}

Response (200 OK):
{
  "success": true,
  "message": "Login successful",
  "data": {
    "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...",
    "user": {
      "id": 123,
      "firstName": "John",
      "lastName": "Doe",
      "email": "john.doe@company.com",
      "role": "EMPLOYEE",
      "companyId": 1
    }
  }
}
```

8.2 Employee Management APIs

Endpoint	Description	Response
GET /api/employees	List all employees (paginated)	Returns employee list
GET /api/employees/{id}	Get employee by ID	Returns employee details
POST /api/employees	Create new employee	Returns created employee
PUT /api/employees/{id}	Update employee	Returns updated employee
DELETE /api/employees/{id}	Delete employee	Returns success message
GET /api/employees/search	Search employees	Returns matching employees

8.3 Time Tracking APIs

Endpoint	Description	Response
POST /api/time-logs/clock-in	Clock in	Returns time log ID
PUT /api/time-logs/clock-out	Clock out	Returns updated time log
GET /api/time-logs	List time logs (filtered)	Returns time logs
GET /api/time-logs/{id}	Get time log by ID	Returns time log details
GET /api/time-logs/employee/{employeeId}	Get employee time logs	Returns time logs
GET /api/time-logs/report	Generate time report	Returns aggregated data

IX. Database Schema

9.1 Entity Relationship Diagram

The TeamSphere database schema implements a normalized relational model with proper foreign key constraints, indexes, and referential integrity. The schema supports multi-tenancy through the company_id discriminator present in all tenant-aware tables.

<Insert Entity Relationship Diagram Here>

Figure 2: Complete entity relationship diagram showing all tables, relationships, and cardinalities.

9.2 Core Tables

Companies Table:

Column	Data Type	Constraints	Description
id	BIGSERIAL	PRIMARY KEY	Unique company identifier
name	VARCHAR(200)	NOT NULL	Company name
domain	VARCHAR(100)	UNIQUE	Company domain (e.g., acme.teamsphere.com)
subscription_plan	VARCHAR(50)		SaaS plan: BASIC, PRO, ENTERPRISE
subscription_status	VARCHAR(20)		ACTIVE, SUSPENDED, CANCELLED
max_employees	INTEGER		Employee limit based on plan
created_at	TIMESTAMP	NOT NULL	Company creation timestamp
updated_at	TIMESTAMP		Last update timestamp

Users Table:

Column	Data Type	Constraints	Description
id	BIGSERIAL	PRIMARY KEY	Unique user identifier
company_id	BIGINT	NOT NULL, FK	Tenant discriminator
employee_id	BIGINT	FK	Link to employee record
email	VARCHAR(100)	UNIQUE, NOT NULL	Login email
password_hash	VARCHAR(255)	NOT NULL	Bcrypt hashed password
role	VARCHAR(20)	NOT NULL	SUPER_ADMIN, ADMIN, HR, MANAGER, EMPLOYEE
is_active	BOOLEAN	DEFAULT TRUE	Account status
last_login	TIMESTAMP		Last successful login
failed_login_attempts	INTEGER	DEFAULT 0	Failed login counter
created_at	TIMESTAMP	NOT NULL	User creation timestamp

Employees Table:

Column	Data Type	Constraints	Description
id	BIGSERIAL	PRIMARY KEY	Unique employee identifier
company_id	BIGINT	NOT NULL, FK	Tenant discriminator
first_name	VARCHAR(50)	NOT NULL	First name
last_name	VARCHAR(50)	NOT NULL	Last name
email	VARCHAR(100)	UNIQUE	Employee email
phone	VARCHAR(20)		Phone number
department_id	BIGINT	FK	Department reference
position	VARCHAR(100)		Job title
manager_id	BIGINT	FK	Self-referencing manager
hire_date	DATE	NOT NULL	Date of hire
nfc_tag_id	VARCHAR(50)	UNIQUE	NFC badge ID
status	VARCHAR(20)		ACTIVE, ON_LEAVE, TERMINATED
created_at	TIMESTAMP	NOT NULL	Record creation

Time Logs Table (Immutable):

Column	Data Type	Constraints	Description
id	BIGSERIAL	PRIMARY KEY	Unique time log identifier
company_id	BIGINT	NOT NULL, FK	Tenant discriminator
employee_id	BIGINT	NOT NULL, FK	Employee reference
clock_in	TIMESTAMP	NOT NULL	Clock in timestamp (immutable)
clock_out	TIMESTAMP		Clock out timestamp (immutable)
work_duration_minutes	INTEGER		Calculated duration
workday_date	DATE	NOT NULL	Workday attribution (04:00 boundary)
location_lat	DECIMAL(10,8)		GPS latitude
location_lon	DECIMAL(11,8)		GPS longitude
device_fingerprint	VARCHAR(255)		Device signature
nfc_tag_id	VARCHAR(50)		NFC tag used
created_at	TIMESTAMP	NOT NULL	Creation (IMMUTABLE)

9.3 Indexes & Performance Optimization

Strategic indexes are critical for query performance, especially in multi-tenant environments with large datasets. TeamSphere implements over 100 indexes across all tables.

Index Name	Columns	Purpose
idx_employees_company	employees(company_id)	Multi-tenancy filtering
idx_employees_email	employees(email)	Login and search
idx_employees_nfc	employees(nfc_tag_id)	NFC authentication
idx_timelogs_company_date	time_logs(company_id, workday_date)	Date range queries
idx_timelogs_employee	time_logs(employee_id)	Employee time reports
idx_leaves_company_status	leave_requests(company_id, status)	Pending approvals
idx_audit_timestamp	audit_logs(created_at)	Recent activity queries
idx_users_email	users(email)	Login performance

9.4 Migration Strategy

TeamSphere uses Flyway for database version control and migrations. Each migration is versioned, tracked, and applied automatically during application startup.

Migration File	Description
V1__initial_schema.sql	Create all core tables and relationships
V2__add_indexes.sql	Add performance indexes
V3__add_audit_tables.sql	Create audit logging tables
V4__add_nfc_support.sql	Add NFC tag fields and device fingerprinting
V5__add_leave_management.sql	Create leave request tables
V6__add_ticketing_system.sql	Create support ticket tables
V7__add_immutable_constraints.sql	Add triggers to prevent time log updates
V8__add_row_level_security.sql	Enable PostgreSQL RLS policies
V9__add_wiki_tables.sql	Create documentation wiki tables
V10__add_backup_metadata.sql	Create backup tracking tables
V11__performance_optimizations.sql	Additional indexes and partitioning

X. User Roles & Permissions

10.1 Role Hierarchy

TeamSphere implements a hierarchical role system where higher-level roles inherit permissions from lower-level roles, plus additional administrative capabilities.

```
SUPER_ADMIN (Platform Level - SaaS Only)
    └── ADMIN (Company Level)
        └── HR (HR Department)
            └── MANAGER (Department/Team)
                └── EMPLOYEE (Individual Contributor)
```

10.2 Permission Matrix

Comprehensive permission matrix defining access control for all major features across different roles.

Permission	SUPER_ADMIN	ADMIN	HR	MANAGER	EMPLOYEE
View All Employees	✓	✓	✓	Team Only	Self Only
Create Employees	✓	✓	✓	✗	✗
Edit Employees	✓	✓	✓	✗	✗
Delete Employees	✓	✓	✗	✗	✗
View Time Logs	✓	✓	✓	Team Only	Self Only
Create Time Logs	✓	✓	✓	✗	Self Only
Edit Time Logs	✓	✓	✓	✗	✗
Delete Time Logs	✓	✓	✗	✗	✗
Approve Leave Requests	✓	✓	✓	Team Only	✗
Submit Leave Requests	✓	✓	✓	✓	✓
View Reports	✓	✓	✓	Team Only	Self Only
Export Data	✓	✓	✓	✗	✗
Import Data	✓	✓	✓	✗	✗
Manage Departments	✓	✓	✗	✗	✗
Manage Users	✓	✓	✗	✗	✗
System Settings	✓	✓	✗	✗	✗
View Audit Logs	✓	✓	✗	✗	✗
Backup/ Restore	✓	✓	✗	✗	✗

XI. Future Enhancements

11.1 Planned Modules

Module	Target Release	Description
Payroll Integration	Q2 2026	Calculate and process payroll based on time logs
Performance Management	Q3 2026	Goals, reviews, and performance tracking
Recruitment Module	Q4 2026	Applicant tracking and hiring workflows
Training & Development	Q1 2027	Course catalog, certifications, skill tracking
Expense Management	Q2 2027	Expense claims, approvals, and reimbursements
Asset Management	Q3 2027	Track company assets assigned to employees

11.2 Scalability Improvements

- Database sharding for companies with >100,000 employees
- Read replicas for report generation and analytics
- Redis caching layer for frequently accessed data
- Message queue (RabbitMQ/Kafka) for async processing
- Elasticsearch for advanced search capabilities
- GraphQL API for flexible data fetching

11.3 AI & Predictive Analytics

- Attendance anomaly detection (fraud prevention)
- Turnover prediction based on employee patterns
- Optimal scheduling recommendations
- Leave pattern analysis and forecasting
- Chatbot for HR policy questions
- Sentiment analysis from employee feedback

XII. Appendix

Appendix A: Sample API Requests & Responses

Clock-In API Request:

```
POST /api/time-logs/clock-in
Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...
Content-Type: application/json

{
  "employeeId": 123,
  "nfcTagId": "E4:3C:2A:1F:8B:42",
  "location": {
    "latitude": 40.7128,
    "longitude": -74.0060
  },
  "deviceFingerprint": "a3f5c8d9e2b1f4a7c6d8e9f0a1b2c3d4"
}

Response (200 OK):
{
  "success": true,
  "message": "Clocked in successfully",
  "data": {
    "timeLogId": 456,
    "clockIn": "2025-10-23T08:00:00Z",
    "workdayDate": "2025-10-23",
    "location": "40.7128,-74.0060"
  }
}
```

Appendix B: Excel Template Specifications

The employee import template must contain the following columns in exact order:

Column Name	Data Type	Required	Description
first_name	VARCHAR	Required	Employee first name
last_name	VARCHAR	Required	Employee last name
email	VARCHAR	Required	Unique email address
phone	VARCHAR	Optional	Phone number
department	VARCHAR	Required	Department name
position	VARCHAR	Required	Job title
hire_date	DATE	Required	YYYY-MM-DD format
manager_email	VARCHAR	Optional	Manager's email (must exist)
nfc_tag_id	VARCHAR	Optional	NFC badge ID

Appendix C: Glossary of Technical Terms

Term	Definition
API	Application Programming Interface - Interface for software communication
JWT	JSON Web Token - Stateless authentication token
RBAC	Role-Based Access Control - Permission model based on user roles
ORM	Object-Relational Mapping - Database abstraction layer
CRUD	Create, Read, Update, Delete - Basic database operations
SaaS	Software as a Service - Cloud-based software delivery model
Multi-Tenancy	Single application instance serving multiple customers
NFC	Near Field Communication - Short-range wireless technology
GDPR	General Data Protection Regulation - EU data privacy law
HA	High Availability - System design for maximum uptime

Appendix D: Reference Links

Official Documentation:

• Spring Boot: https://spring.io/projects/spring-boot

• React: https://react.dev/

• PostgreSQL: https://www.postgresql.org/docs/

• Docker: https://docs.docker.com/

• Kubernetes: https://kubernetes.io/docs/

Standards & Compliance:

• IEEE 1063: Software User Documentation

• ISO/IEC/IEEE 26511: Requirements for Managers of User Documentation

• GDPR Official Text: https://gdpr-info.eu/

• SOX Compliance: https://www.sarbanes-oxley-101.com/

Document Control

Version	Date	Author	Changes
1.0.0	2025-10-23	TeamSphere Dev Team	Initial release - Complete technical documentation

——— End of Document ———