**Sports Academy Management System**

# **System Architecture Document**

## **Version:** 2.0

## **Document Status:** -

## **Last Updated:** June 2025

**Prepared by:** System Architecture Team

# **Table of Contents**

1. Executive Summary

2. System Architecture Overview

3. Database Design & Entity Relationship Model

4. User Interface Design

5. Core System Workflows

6. Performance & Scalability Strategy

7. Error Handling & Recovery Strategy

8. Security Architecture

9. Deployment Architecture

10. Success Metrics

11. Future Enhancements

# **1. Executive Summary**

## **1.1 Project Overview**

This document presents the comprehensive system architecture for a scalable sports academy management platform designed to serve **127+ students**, **4 coaches**, and **1 head coach** across multiple training facilities. The system delivers integrated solutions for attendance tracking, performance management, and fee administration through a responsive, role-based web interface.

## **1.2 Key Architectural Principles**

• **Microservices Architecture**: Modular, independently scalable services

• **Cloud-Native Design**: Container-based deployment with orchestration

• **Mobile-First Responsive UI**: Optimized for all device types

• **Real-Time Data Synchronization**: Instant updates across all user interfaces

• **Role-Based Access Control**: Secure, permission-based system access

• **High Availability**: 99.9% uptime with automated failover capabilities

• **Enhanced coach productivity** through integrated performance tools

# **2. System Architecture Overview**

## **2.1 Three-Tier Architecture Design**

The system employs a modern three-tier architecture pattern optimized for scalability and maintainability:

PRESENTATION TIER

React.js SPA • Material-UI • Progressive Web App Mobile Responsive • Real-time Updates • Role-based Views

HTTPS/WebSocket

APPLICATION TIER

API Gateway • Microservices • Business Logic

Auth • Attendance • Performance • Fees • Notifications

Database Connections

DATA TIER

PostgreSQL • Redis Cache • AWS S3 • Backup Systems

ACID Compliance • High Availability • Encryption

## **2.2 Microservices Architecture Core Services Overview**

1. **Authentication Service** - JWT-based security and user management

2. **Attendance Management Service** - Session tracking and cross-facility operations

3. **Performance Tracking Service** - Skills assessment and progress analytics

4. **Fee Management Service** - Payment tracking and financial reporting

5. **User Management Service** - Profile and role administration

6. **Notification Service** - Real-time alerts and communications

7. **Reporting Service** - Analytics and dashboard data generation

## **Service Communication Pattern**

API Gateway (Load Balancer)

Authentication Service

Attendance Service ←→ User Service

Performance Service ←→ User Service

Fee Service ←→ Notification Service

User Management Service

Notification Service ←→ All Services

# **3. Database Design & Entity Relationship Model**

## **3.1 Entity Relationship Diagram**

erDiagram

USERS { uuid id PK varchar username UK varchar email UK varchar password\_hash enum role enum status timestamp created\_at timestamp updated\_at

}

FACILITIES { uuid id PK varchar name varchar code UK text address jsonb settings timestamp created\_at

}

STUDENTS { uuid id PK uuid user\_id FK varchar student\_id UK varchar first\_name varchar last\_name uuid primary\_facility\_id FK varchar current\_position date enrollment\_date

enum status jsonb emergency\_contacts

}

COACHES { uuid id PK uuid user\_id FK varchar coach\_id UK varchar first\_name varchar last\_name varchar specialization enum status jsonb certifications

}

SESSIONS { uuid id PK uuid facility\_id FK uuid coach\_id FK enum session\_type varchar batch\_name timestamp start\_time timestamp end\_time enum status jsonb session\_config

}

ATTENDANCE\_RECORDS { uuid id PK uuid student\_id FK uuid session\_id FK enum attendance\_type enum status uuid marked\_by FK timestamp marked\_at text notes

}

PERFORMANCE\_ASSESSMENTS { uuid id PK uuid student\_id FK uuid coach\_id FK uuid session\_id FK date assessment\_date jsonb technical\_scores jsonb tactical\_scores jsonb physical\_scores jsonb psychological\_scores text coach\_notes timestamp created\_at

}

FEE\_STRUCTURES { uuid id PK uuid facility\_id FK varchar plan\_name decimal monthly\_fee decimal registration\_fee jsonb additional\_charges boolean is\_active

}

FEE\_RECORDS { uuid id PK uuid student\_id FK uuid fee\_structure\_id FK decimal amount\_due decimal amount\_paid date due\_date date paid\_date enum payment\_status varchar payment\_method text notes uuid recorded\_by FK

}

FACILITY\_ASSIGNMENTS { uuid id PK uuid student\_id FK uuid facility\_id FK date start\_date date end\_date boolean is\_active }

SESSION\_ENROLLMENTS { uuid id PK uuid student\_id FK uuid session\_id FK date enrollment\_date enum status

}

%% Relationships

USERS ||--o{ STUDENTS : "user\_id"

USERS ||--o{ COACHES : "user\_id"

FACILITIES ||--o{ STUDENTS : "primary\_facility\_id"

FACILITIES ||--o{ SESSIONS : "facility\_id"

FACILITIES ||--o{ FEE\_STRUCTURES : "facility\_id"

COACHES ||--o{ SESSIONS : "coach\_id"

COACHES ||--o{ PERFORMANCE\_ASSESSMENTS : "coach\_id"

STUDENTS ||--o{ ATTENDANCE\_RECORDS : "student\_id"

STUDENTS ||--o{ PERFORMANCE\_ASSESSMENTS : "student\_id"

STUDENTS ||--o{ FEE\_RECORDS : "student\_id"

STUDENTS ||--o{ FACILITY\_ASSIGNMENTS : "student\_id"

STUDENTS ||--o{ SESSION\_ENROLLMENTS : "student\_id"

SESSIONS ||--o{ ATTENDANCE\_RECORDS : "session\_id"

SESSIONS ||--o{ PERFORMANCE\_ASSESSMENTS : "session\_id"

SESSIONS ||--o{ SESSION\_ENROLLMENTS : "session\_id"

FACILITIES ||--o{ FACILITY\_ASSIGNMENTS : "facility\_id"

FEE\_STRUCTURES ||--o{ FEE\_RECORDS : "fee\_structure\_id"

## **3.2 Database Schema Details**

### **3.2.1 Core Tables Structure USERS Table**

**CREATE TABLE** users ( **id** UUID **PRIMARY KEY DEFAULT** gen\_random\_uuid(), username VARCHAR(50) **UNIQUE NOT NULL**, email VARCHAR(100) **UNIQUE NOT NULL**, password\_hash VARCHAR(255) **NOT NULL**, **role** user\_role **NOT NULL**, status user\_status **DEFAULT** 'active', last\_login TIMESTAMP, created\_at TIMESTAMP **DEFAULT** NOW(), updated\_at TIMESTAMP **DEFAULT** NOW()

);

**CREATE TYPE** user\_role **AS** ENUM ('student', 'coach', 'head\_coach');

**CREATE TYPE** user\_status **AS** ENUM ('active', 'inactive', 'archived');

## **STUDENTS Table**

**CREATE TABLE** students ( **id** UUID **PRIMARY KEY DEFAULT** gen\_random\_uuid(), user\_id UUID **REFERENCES** users(**id**) **ON DELETE CASCADE**, student\_id VARCHAR(20) **UNIQUE NOT NULL**, first\_name VARCHAR(50) **NOT NULL**, last\_name VARCHAR(50) **NOT NULL**,

primary\_facility\_id UUID **REFERENCES** facilities(**id**), current\_position VARCHAR(30), enrollment\_date DATE **NOT NULL**, date\_of\_birth DATE, emergency\_contacts JSONB, medical\_info JSONB, status student\_status **DEFAULT** 'active', created\_at TIMESTAMP **DEFAULT** NOW()

);

**CREATE TYPE** student\_status **AS** ENUM ('active', 'inactive', 'transferred', 'graduated');

## **3.3 Indexing Strategy Performance Optimization Indexes**

*-- User authentication and lookup*

**CREATE INDEX** idx\_users\_email **ON** users(email);

**CREATE INDEX** idx\_users\_username **ON** users(username);

**CREATE INDEX** idx\_users\_role\_status **ON** users(**role**, status);

*-- Attendance queries optimization*

**CREATE INDEX** idx\_attendance\_student\_date **ON** attendance\_records(student\_id, marked\_at);

**CREATE INDEX** idx\_attendance\_session **ON** attendance\_records(session\_id);

**CREATE INDEX** idx\_attendance\_status **ON** attendance\_records(status);

*-- Performance tracking*

**CREATE INDEX** idx\_performance\_student\_date **ON** performance\_assessments(student\_id, assessment\_ **CREATE INDEX** idx\_performance\_coach **ON** performance\_assessments(coach\_id);

*-- Fee management*

**CREATE INDEX** idx\_fee\_records\_student\_status **ON** fee\_records(student\_id, payment\_status);

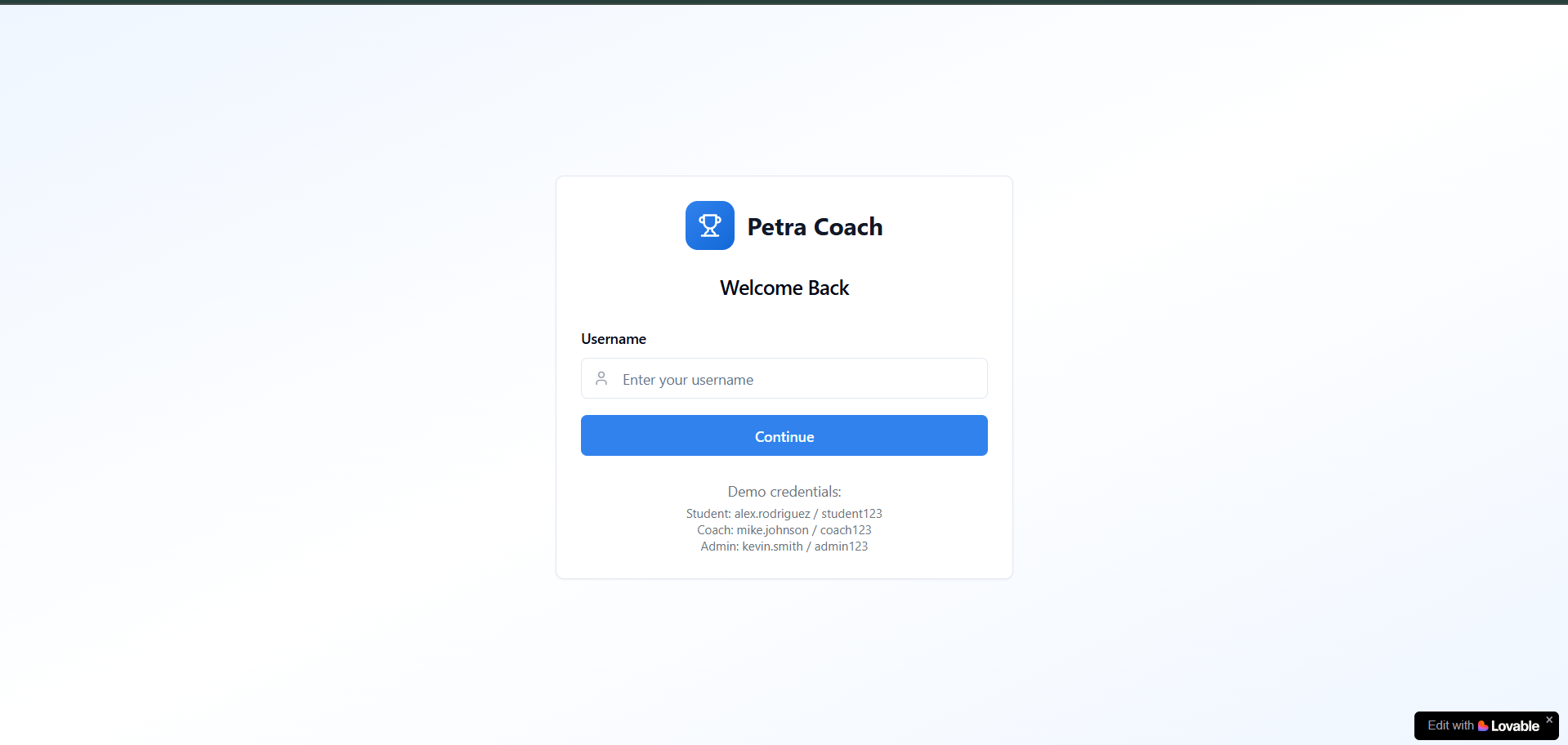
**CREATE INDEX** idx\_fee\_records\_due\_date **ON** fee\_records(due\_date) **WHERE** payment\_status = 'pendi

*-- Session management*

**CREATE INDEX** idx\_sessions\_facility\_date **ON** sessions(facility\_id, start\_time);

**CREATE INDEX** idx\_sessions\_coach\_date **ON** sessions(coach\_id, start\_time);

# **4. User Interface Design**

**LOGIN PAGE:   
  
**

The login page for the website first asks for our username, on the basis of the username it judges on what kind of user you are. If you are a coach it opens the coach password page.

|  |  |  |  |
| --- | --- | --- | --- |
|  |  | |  |
|  |  | |  |
| **ADMIN PAGE:   Home page:** |  | |  |
| The home page includes a quick overview of total students, upcoming sessions, overview window and recent activity.  *Upcoming sessions:*  This includes what all sessions are going to be held on that particular day, it gets updated with time as the  sessions all have their assigned time.  *Overview window:* Would show us the total number of institutes, students, and total number of coaches.  *Recent activity(optional):* Would show us the sessions that have ended, any updates that have been made to the website, such as fee updates, attendance updates etc.  **Students page:**    The student page for the admin is to have a window called the students overview.  *Students overview:*   * All students: This includes the list of all the students in the institute, they can be filtered on the basis of institute, session and coaches. It contains the general information regarding the students. (Names, centre, contact information). * Add student: The admin can add new students in this section, it would contain Student information(Name, contact info, training etc.). It would also contain training details, which will assign the students to their respective classes.   **Coaches page:** |  | |  |
| The coaches page, aims to handle the coaches and their assignments. We have to make sure that these coaches are assigned to the right training session.  *Management overview:*   * Coach overview: This will show us all the coaches which are presently working in the institute. * Coach management: We can add and remove coaches here, also change the sessions assigned to the coaches. It should be divided into multiple parts: Add coach, remove and edit coach(we can change the sessions that have been assigned to the coach).     **Sessions page:** |  | |  |
| In the session page, we aim to keep track of all the sessions and be able to edit these sessions.  *Session overview:*   * All sessions: This is a list of all the sessions that are currently available in the institute across all the centres. * Session management: We can add and remove sessions in this section, where we can add new sessions across institutes and remove sessions too.   **Attendance page:** |  | |  |
|  |  | |  |
|  |  | |  |
| In this page, we aim to keep track of attendance of students and also for the admin to mark attendance as the attendance has a complete overview of the institute. He is able to change attendance of all students in the institutes unlike the coach where the coach can mark attendance only for his assigned sessions.  *Attendance management:*   * Attendance reports: attendance of all the students is displayed, it's divided by campus which have further subdivisions of sessions. * Mark attendance: Here the admin can edit attendance of students. |  | |  |
| **Progress page** |  | |  |
|  |  | |  |
|  |  | |  |
|  |  | |  |
| This page is for the admin to give the progress scores of the students.  *Progress tracking:*   * Progress overview: Progress scores of all the students will be shown here. The details that will be showcased are the name, assigned position, (performance metrics), overall, date of assessment. * New assessment: The admin is going to make a new assessment from here. Where progress is recorded. Admin chooses a student from the student roster and gives them their assessment. * Progress history: Here the admin is able to look at previous progress scores of students. He can search for a student and the student's previous scores will be given.   **Fees page:** |  | |  |
| Fees page is to show the admin which all students have paid and are yet to pay.  *Quick overview: Unpaid students(number of unpaid students)*  *Fee management:*   * All students: It’s a list of all students and their fee status. * Unpaid students: It gives us a list of all the students who have not yet paid. * Update: This is where the admin can update the fee payment status of all students. You can search for students and |  | |  |
|  | |  | |
|  | |  | |
|  | |  | |
|  | |  | |
|  | |  | |
|  | |  | |
|  | |  | |
|  | |  | |
|  | |  | |
|  | |  | |
|  | |  | |
|  | |  | |
|  | |  | |
|  | |  | |
|  | |  | |
|  | |  | |
|  | |  | |
|  | |  | |
|  | |  | |
|  | |  | |
|  | |  | |
|  |  |  |  |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | |  | | | |
|  | |  | |  | |
|  | |  | |  | |
|  | |  | |  | |
|  |  |  | |  | |
|  |  |  | |  | |
|  |  |  | |  | |
|  |  |  | |  | |
|  |  |  | |  | |
|  |  |  | |  | |
|  |  |  | |  | |
|  |  |  | |  | |
|  |  |  | |  | |
|  |  |  | |  | |
|  |  |  | |  | |
|  |  |  | |  | |
|  |  |  | |  | |
|  |  |  | |  | |
|  |  |  | |  | |
|  |  |  | |  | |
|  |  |  | |  | |
|  |  |  | |  | |
|  | |  | |  |  |
|  | |  | |  |  |
|  | |  | | |  |
|  | |  | | |  |
|  | |  | | |  |
|  | |  | | |  |
|  | |  | | |  |
|  | |  | | |  |
|  | |  | | |  |
|  | |  | | |  |
|  | |  | | | |
|  | | |  | | |
|  | | |  | | |
|  | | |  | | |
|  | | |  | | |
|  | | |  | | |
|  | | |  | | |
|  | | |  | | |
|  | | |  | | |
|  | | |  | | |
|  | | |  | | |
|  | | |  | | |
|  |  |  |  |  |  |

# **6. Performance & Scalability Strategy**

## **6.1 Load Balancing Architecture**

### **6.1.1 Multi-Tier Load Balancing**

Internet Traffic

↓

LEVEL 1: DNS Load Balancing

Route 53 / CloudFlare DNS (Geographic)

↓

LEVEL 2: Application Load Balancer

NGINX/HAProxy (Round Robin + Health Checks)

Region A Region B Region C

Active-Active Active-Active Active-Standby

↓

LEVEL 3: Service Load Balancer

Kubernetes Ingress Controller

Auth Service API Services Worker Queue

(3 pods) (5 pods) (2 pods)

**6.1.2 Load Balancing Configuration NGINX Load Balancer Configuration:**

upstream sports\_academy\_backend { least\_conn;

server backend1.academy.com:3000 weight=3 max\_fails=3 fail\_timeout=30s; server backend2.academy.com:3000 weight=3 max\_fails=3 fail\_timeout=30s; server backend3.academy.com:3000 weight=2 max\_fails=3 fail\_timeout=30s; keepalive 32;

}

server { listen 80; listen 443 ssl http2;

# Health check endpoint location /health { access\_log off;

proxy\_pass http://sports\_academy\_backend/health; proxy\_set\_header Host $host;

}

# Main application location / { proxy\_pass http://sports\_academy\_backend; proxy\_set\_header Host $host; proxy\_set\_header X-Real-IP $remote\_addr;

proxy\_set\_header X-Forwarded-For $proxy\_add\_x\_forwarded\_for; proxy\_connect\_timeout 5s; proxy\_send\_timeout 60s; proxy\_read\_timeout 60s;

}

}

## **6.2 Horizontal Scaling Strategy**

**6.2.1 Auto-Scaling Configuration Kubernetes Horizontal Pod Autoscaler:**

apiVersion**:** autoscaling/v2 kind**:** HorizontalPodAutoscaler metadata**:** name**:** sports-academy-api

spec**:**

scaleTargetRef**:** apiVersion**:** apps/v1 kind**:** Deployment name**:** sports-academy-api

minReplicas**:** 3 maxReplicas**:** 20 metrics**:**

- type**:** Resource resource**:**

name**:** cpu target**:** type**:** Utilization averageUtilization**:** 70

- type**:** Resource resource**:** name**:** memory target**:** type**:** Utilization averageUtilization**:** 80

behavior**:**

scaleUp**:** stabilizationWindowSeconds**:** 60 policies**:**

- type**:** Percent value**:** 100 periodSeconds**:** 15

scaleDown**:** stabilizationWindowSeconds**:** 300 policies**:**

- type**:** Percent value**:** 50 periodSeconds**:** 60

**6.2.2 Database Scaling Strategy Read Replica Configuration:**

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
| DATABASE SCALING |  |  |  |
|  |  |  |  |
| WRITE OPERATIONS READ OPERATIONS |  |  |  |
| Master DB Read Replica |  |  |  |
| (PostgreSQL) Pool (3x) |  |  |  |
| Write-Heavy Read-Heavy |  |  |  |
| - Attendance - Dashboards |  |  |  |
| - Payments - Reports |  |  |  |
| - Assessments - Analytics |  |  |  |
|  |  |  |  |
| CONNECTION POOLING |  |  |  |
| PgBouncer Pool Manager |  |  | |
| - Max Connections: 200 |  |  | |
| - Pool Size: 25 per service |  |  | |
| - Connection Timeout: 30s |  |  | |

## **6.3 Caching Strategy Implementation**

### **6.3.1 Multi-Level Caching Architecture**

CACHING LAYERS

LEVEL 1: Browser Cache (Client-Side)

• Static Assets: 1 year

• API Responses: 5 minutes

• User Preferences: Session

↓

LEVEL 2: CDN Cache (Edge Locations)

• Images/Videos: 24 hours

• JavaScript/CSS: 1 hour

• Public API Data: 15 minutes

↓

LEVEL 3: Application Cache (Redis)

• Session Data: 24 hours

• Dashboard Metrics: 10 minutes

• User Profiles: 1 hour

• Attendance Stats: 5 minutes

↓

LEVEL 4: Database Query Cache

• Frequent Queries: 30 minutes

• Report Data: 2 hours

• Reference Data: 12 hours

**6.3.2 Redis Cache Implementation Cache Configuration:**

*// Redis Cache Service* **class** CacheService {

constructor() {

**this**.redis = **new** Redis({

host: process.env.REDIS\_HOST, port: process.env.REDIS\_PORT, password: process.env.REDIS\_PASSWORD, retryDelayOnFailover: 100, maxRetriesPerRequest: 3, lazyConnect: **true**

});

}

*// Dashboard metrics caching*

**async** cacheDashboardMetrics(userId, role, data) {

**const** key = `dashboard:${role}:${userId}`; **await this**.redis.setex(key, 600, JSON.stringify(data)); *// 10 minutes*

}

*// Attendance statistics caching* **async** cacheAttendanceStats(facilityId, data) {

**const** key = `attendance:stats:${facilityId}`; **await this**.redis.setex(key, 300, JSON.stringify(data)); *// 5 minutes*

}

*// Session management*

**async** cacheUserSession(sessionId, userData) {

**const** key = `session:${sessionId}`; **await this**.redis.setex(key, 86400, JSON.stringify(userData)); *// 24 hours* }

}

## **6.4 Performance Monitoring & Optimization**

**6.4.1 Performance Metrics Tracking Key Performance Indicators:**

|  |  |
| --- | --- |
|  |  |
| PERFORMANCE MONITORING |  |
|  |  |
| RESPONSE TIME TARGETS |  |
| Dashboard Load Time: < 1.5 seconds |  |
| API Response Time: < 300ms (95th percentile) |  |
| Database Query Time: < 100ms (average) |  |
| Real-time Updates: < 50ms latency |  |
|  |  |
| THROUGHPUT TARGETS |  |
| Concurrent Users: 500+ simultaneous |  |
| API Requests: 10,000 requests/minute |  |
| Database Connections: 200 max concurrent |  |

|  |  |  |  |
| --- | --- | --- | --- |
| Memory Usage: < 80% per container |  | | |
|  |  | | |
| AVAILABILITY TARGETS |  | | |
| System Uptime: 99.9% (8.76 hours downtime/year) |  | | |
| Recovery Time: < 5 minutes (RTO) |  | | |
| Data Loss: < 1 minute (RPO)    **7.** **Error Handling & Recovery Strategy**  **7.1** **Comprehensive Error Handling Framework**  **7.1.1 Error Classification System** |  | | |
| ERROR CATEGORIES |  | | |
|  |  | | |
| CRITICAL ERRORS (System Down) |  | | |
| • Database connection failures |  | | |
| • Authentication service unavailable |  | | |
| • Complete system outage |  | | |
| Response: Immediate escalation + failover |  | | |
|  |  | | |
| HIGH PRIORITY ERRORS (Feature Impact) |  | | |
| • Payment processing failures |  | | |
| • Attendance marking errors |  | | |
| • Performance data corruption |  | | |
| Response: Auto-retry + alert + manual intervention |  | | |
|  |  | | |
| MEDIUM PRIORITY ERRORS (User Experience) |  | | |
| • Dashboard loading delays |  | | |
| • Report generation timeouts |  | | |
| • Notification delivery failures |  | | |
| Response: Graceful degradation + user notification |  | | |
|  |  | | |
| LOW PRIORITY ERRORS (Non-Critical) |  | | |
|  | |  |  |
| • UI rendering glitches | |  |  |
| • Non-essential feature failures | |  |  |
| • Cosmetic issues | |  |  |
| Response: Log + queue for next release | |  |  |
|  |  |  |  |

**7.1.2 Error Handling Implementation Global Error Handler:**

*// Global Error Handling Middleware* **class** ErrorHandler {

constructor() {

**this**.logger = **new** Logger('ErrorHandler'); **this**.alerting = **new** AlertingService();

}

**async** handleError(error, context) {

**const** errorInfo = **this**.classifyError(error);

*// Log error with context* **await this**.logError(errorInfo, context);

*// Determine response strategy* **switch**(errorInfo.severity) {

**case** 'CRITICAL': **await this**.handleCriticalError(errorInfo, context); **break**;

**case** 'HIGH': **await this**.handleHighPriorityError(errorInfo, context); **break**;

**case** 'MEDIUM':

**await this**.handleMediumPriorityError(errorInfo, context); **break**;

**case** 'LOW': **await this**.handleLowPriorityError(errorInfo, context); **break**;

}

}

**async** handleCriticalError(errorInfo, context) {

*// Immediate escalation*

**await this**.alerting.sendCriticalAlert(errorInfo);

*// Attempt automatic failover* **await this**.initiateFailover(context); *// Notify all stakeholders*

**await this**.notifyStakeholders(errorInfo);

}

**async** handleHighPriorityError(errorInfo, context) { *// Auto-retry with exponential backoff*

**const** retryResult = **await this**.retryWithBackoff(context.operation, 3);

**if** (!retryResult.success) { *// Alert operations team*

**await this**.alerting.sendHighPriorityAlert(errorInfo);

*// Enable graceful degradation*

**await this**.enableGracefulDegradation(context.feature); }

}

}

*// Retry mechanism with exponential backoff* **class** RetryService {

**async** retryWithBackoff(operation, maxRetries = 3, baseDelay = 1000) {

**for** (**let** attempt = 1; attempt <= maxRetries; attempt++) {

**try** {

**const** result = **await** operation(); **return** { success: **true**, result };

} **catch** (error) { **if** (attempt === maxRetries) {

**return** { success: **false**, error, attempts: attempt };

}

**const** delay = baseDelay \* Math.pow(2, attempt - 1);

**await this**.sleep(delay);

}

}

}

sleep(ms) {

**return new** Promise(resolve **=>** setTimeout(resolve, ms)); }

}

## **7.2 Circuit Breaker Pattern Implementation**

*// Circuit Breaker for External Dependencies* **class** CircuitBreaker {

constructor(options = {}) {

**this**.failureThreshold = options.failureThreshold || 5; **this**.timeout = options.timeout || 60000; **this**.monitoringPeriod = options.monitoringPeriod || 10000;

**this**.state = 'CLOSED'; *// CLOSED, OPEN, HALF\_OPEN* **this**.failureCount = 0; **this**.lastFailureTime = **null**;

}

**async** execute(operation) {

**if** (**this**.state === 'OPEN') { **if** (Date.now() - **this**.lastFailureTime >= **this**.timeout) {

**this**.state = 'HALF\_OPEN';

} **else** { **throw new** Error('Circuit breaker is OPEN');

}

}

**try** {

**const** result = **await** operation();

**this**.onSuccess(); **return** result;

} **catch** (error) {

**this**.onFailure(); **throw** error;

}

}

onSuccess() {

**this**.failureCount = 0; **this**.state = 'CLOSED';

}

onFailure() {

**this**.failureCount++; **this**.lastFailureTime = Date.now();

**if** (**this**.failureCount >= **this**.failureThreshold) {

**this**.state = 'OPEN';

}

}

}

## **7.3 Data Recovery & Backup Strategy**

## **7.3.1 Automated Backup System**

|  |  |
| --- | --- |
|  |  |
| BACKUP STRATEGY |  |
|  |  |
| BACKUP TIERS |  |
| Tier 1: Real-time Replication |  |
| • Master-Slave PostgreSQL |  |
| • Continuous WAL shipping |  |
| • Recovery Point: 0-1 minutes |  |
|  |  |
| Tier 2: Hourly Snapshots |  |
| • Database point-in-time recovery |  |
| • Application state backups |  |
| • Recovery Point: 1 hour |  |
|  |  |
| Tier 3: Daily Full Backups |  |
| • Complete system backup |  |
| • Cross-region replication |  |
| • 30-day retention policy |  |
|  |  |
| Tier 4: Weekly Archive Backups |  |
| • Long-term storage (1 year) |  |
| • Compliance and audit trail |  |
| • Cold storage optimization |  |

|  |  |  |  |
| --- | --- | --- | --- |
| Scenario | RTO Target | RPO Target | Recovery Strategy |
| **Database**  **Failure** | 5 minutes | 1 minute | Automatic failover to read replica |
| **Application Server** | 2 minutes | 0 seconds | Load balancer redirect to healthy instances |

**7.3.2 Disaster Recovery Plan Recovery Time Objectives (RTO) & Recovery Point Objectives (RPO):**

## **Down**

|  |  |  |  |
| --- | --- | --- | --- |
| Scenario | RTO Target | RPO Target | Recovery Strategy |
| **Complete**  **Data**  **Center**  **Outage** | 30 minutes | 5 minutes | Cross-region failover with backup restoration |
| **Data**  **Corruption** | 1 hour | 1 hour | Point-in-time recovery from backups |
| **Security Breach** | 4 hours | 15 minutes | Isolated recovery with security patches |

## **7.4 Monitoring & Alerting System**

### **7.4.1 Comprehensive Monitoring Stack**

|  |  |  |  |
| --- | --- | --- | --- |
|  |  | | |
| MONITORING ARCHITECTURE |  | | |
|  |  | | |
| APPLICATION MONITORING |  | | |
| Prometheus + Grafana |  | | |
| • Custom business metrics |  | | |
| • Performance counters |  | | |
| • User behavior analytics |  | | |
| ↓ |  | | |
| INFRASTRUCTURE MONITORING |  | | |
| Node Exporter + cAdvisor |  | | |
| • CPU, Memory, Disk I/O |  | | |
| • Network performance |  | | |
| • Container resource usage |  | | |
| ↓ |  | | |
| LOG AGGREGATION |  | | |
| ELK Stack (Elasticsearch, Logstash, Kibana) |  | | |
| • Centralized log collection |  | | |
| • Real-time log analysis |  | | |
| • Error pattern detection |  | | |
| ↓ |  | | |
| ALERTING & NOTIFICATION |  | | |
| AlertManager + PagerDuty |  | | |
| • Smart alert routing |  | | |
| • Escalation policies | |  |  |
| • Multi-channel notifications | |  |  |
|  |  |  |  |

**7.4.2 Alert Configuration Critical System Alerts:**

*# Prometheus Alert Rules* groups**:**

- name**:** sports\_academy\_critical rules**:**

- alert**:** DatabaseDown expr**:** up{job="postgresql"} == 0 for**:** 30s labels**:** severity**:** critical

annotations**:**

summary**:** "Database is down"

description**:** "PostgreSQL database has been down for more than 30 seconds"

- alert**:** HighErrorRate expr**:** rate(http\_requests\_total{status=~"5.."}[5m]) > 0.1 for**:** 2m labels**:** severity**:** warning

annotations**:**

summary**:** "High error rate detected"

description**:** "Error rate is {{ $value }} errors per second"

- alert**:** AttendanceServiceDown expr**:** up{job="attendance-service"} == 0

for**:** 1m labels**:**

severity**:** high

annotations**:**

summary**:** "Attendance service unavailable" description**:** "Attendance marking functionality is down"

- alert**:** FeePaymentFailures expr**:** increase(fee\_payment\_errors\_total[10m]) > 5 for**:** 0s labels**:**

severity**:** high

annotations**:**

summary**:** "Multiple fee payment failures"

description**:** "{{ $value }} fee payment failures in the last 10 minutes"

# **8. Security Architecture**

## **8.1 Multi-Layer Security Framework**

### **8.1.1 Security Architecture Overview**

|  |  |
| --- | --- |
|  |  |
| SECURITY LAYERS |  |
|  |  |
| LAYER 1: PERIMETER SECURITY |  |
| • Web Application Firewall (WAF) |  |
| • DDoS Protection |  |
| • Rate Limiting |  |
| • IP Whitelisting for Admin Access |  |
| ↓ |  |
| LAYER 2: APPLICATION SECURITY |  |
| • JWT Token Authentication |  |
| • Role-Based Access Control (RBAC) |  |
| • Input Validation & Sanitization |  |
| • CSRF Protection |  |
| ↓ |  |
| LAYER 3: DATA SECURITY |  |
| • Encryption at Rest (AES-256) |  |
| • Encryption in Transit (TLS 1.3) |  |
| • Database Access Controls |  |
| • PII Data Masking |  |
| ↓ |  |
| LAYER 4: INFRASTRUCTURE SECURITY |  |
| • Container Security Scanning |  |
| • Network Segmentation |  |
| • Secrets Management |  |
| • Audit Logging |  |

### **8.1.2 Role-Based Access Control Matrix**

|  |  |  |  |
| --- | --- | --- | --- |
| Feature/Resource | Student | Coach | Head Coach |
| **Dashboard View** | Own data  only | Assigned students | All facilities |
| **Attendance Marking** |  | Own  sessions | All sessions |
| **Performance Assessment** | View only | Own  students | All students |
| **Fee Payment Recording** |  |  | All students |
| **Fee Status View** | Own status |  | All students |
| **User Management** |  |  | Full access |
| **System Settings** |  |  | Full access |
| **Reports Generation** | Own reports | Student reports | All reports |
| **Facility Management** |  |  | Full access |

## **8.2 Authentication & Authorization 8.2.1 JWT Token Implementation**

*// JWT Security Configuration* **class** AuthenticationService {

constructor() {

**this**.jwtSecret = process.env.JWT\_SECRET; **this**.tokenExpiry = '24h'; **this**.refreshTokenExpiry = '7d';

}

generateTokens(user) { **const** payload = { userId: user.id, role: user.role, facilityAccess: user.facilityAccess, iat: Math.floor(Date.now() / 1000)

};

**const** accessToken = jwt.sign(payload, **this**.jwtSecret, {

expiresIn: **this**.tokenExpiry, issuer: 'sports-academy', audience: 'academy-users'

});

**const** refreshToken = jwt.sign( { userId: user.id, type: 'refresh' }, **this**.jwtSecret,

{ expiresIn: **this**.refreshTokenExpiry }

);

**return** { accessToken, refreshToken };

}

verifyToken(token) {

**return** jwt.verify(token, **this**.jwtSecret, {

issuer: 'sports-academy', audience: 'academy-users'

});

}

}

*// Authorization Middleware* **class** AuthorizationMiddleware { **static** requireRole(allowedRoles) {

**return** (req, res, next) **=>** {

**const** user = req.user;

**if** (!allowedRoles.includes(user.role)) { **return** res.status(403).json({

error: 'Insufficient permissions', required: allowedRoles, current: user.role

}); }

next();

}; }

**static** requireFeeManagementAccess() {

**return** (req, res, next) **=>** {

**if** (req.user.role !== 'head\_coach') {

**return** res.status(403).json({

error: 'Fee management access restricted to head coach only' }); } next();

};

}

}

## **8.3 Data Protection & Privacy 8.3.1 Encryption Strategy**

*// Data Encryption Service* **class** EncryptionService {

constructor() {

**this**.algorithm = 'aes-256-gcm'; **this**.keyDerivation = 'pbkdf2';

}

*// Encrypt sensitive data (PII, financial)* encryptSensitiveData(data, context = '') {

**const** key = **this**.deriveKey(context); **const** iv = crypto.randomBytes(16);

**const** cipher = crypto.createCipher(**this**.algorithm, key, iv);

**let** encrypted = cipher.update(JSON.stringify(data), 'utf8', 'hex'); encrypted += cipher.final('hex'); **const** authTag = cipher.getAuthTag();

**return** {

encrypted, iv: iv.toString('hex'), authTag: authTag.toString('hex')

};

}

*// Field-level encryption for PII* encryptPIIFields(studentData) {

**const** sensitiveFields = ['email', 'phone', 'address', 'emergencyContact']; **const** encrypted = { ...studentData };

sensitiveFields.forEach(field **=>** {

**if** (encrypted[field]) {

encrypted[field] = **this**.encryptField(encrypted[field]); } });

**return** encrypted;

}

}

# **9. Deployment Architecture**

## **9.1 Containerized Deployment Strategy**

### **9.1.1 Kubernetes Deployment Configuration**

*# Production Deployment Manifest* apiVersion**:** apps/v1 kind**:** Deployment metadata**:** name**:** sports-academy-api namespace**:** production labels**:** app**:** sports-academy tier**:** backend

spec**:** replicas**:** 5 strategy**:** type**:** RollingUpdate rollingUpdate**:** maxSurge**:** 2 maxUnavailable**:** 1

selector**:**

matchLabels**:** app**:** sports-academy-api template**:**

metadata**:**

labels**:** app**:** sports-academy-api

spec**:** containers**:**

- name**:** api-server image**:** sports-academy/api:v2.0.0 ports**: -** containerPort**:** 3000 env**:**

- name**:** NODE\_ENV value**:** "production"

- name**:** DATABASE\_URL valueFrom**:**

secretKeyRef**:** name**:** database-credentials key**:** url

- name**:** REDIS\_URL valueFrom**:**

secretKeyRef**:**

name**:** redis-credentials

key**:** url

resources**:**

requests**:** memory**:** "256Mi" cpu**:** "250m"

limits**:**

memory**:** "512Mi" cpu**:** "500m"

livenessProbe**:**

httpGet**:** path**:** /health port**:** 3000

initialDelaySeconds**:** 30 periodSeconds**:** 10

readinessProbe**:**

httpGet**:** path**:** /ready port**:** 3000

initialDelaySeconds**:** 5 periodSeconds**:** 5

--apiVersion**:** v1 kind**:** Service metadata**:** name**:** sports-academy-api-service namespace**:** production

spec**:**

selector**:** app**:** sports-academy-api

ports**:**

**-** port**:** 80 targetPort**:** 3000

type**:** ClusterIP

## **9.2 CI/CD Pipeline**

**9.2.1 GitHub Actions Workflow** “‘yaml # .github/workflows/deploy.yml name: Deploy Sports Academy

on: push: branches: [main] pull\_request: branches: [main] jobs: test: runs-on: ubuntu-latest steps: - uses: actions/checkout@v3

- name: Setup Node.js uses: actions/setup-node@v3 with:

node-version: '18'

cache: 'npm'

- name: Install dependencies run: npm ci

- name: Run tests run: npm run test:coverage

- name: Run security audit run: npm audit --audit-level high

build: needs: test runs-on: ubuntu-latest if: github.ref == ‘refs/heads/main’ steps:

- uses: actions/checkout@v3

- name: Build Docker image run: |

docker build -t sports-academy/api:${{ github.sha }} . docker tag sports-academy/api:${{ github.sha }} sports-academy/api:latest

- name: Push to registry run: |

echo ${{ secrets.DOCKER\_PASSWORD }} | docker login -u ${{ secrets.DOCKER\_USERNAME }} --p docker push sports-academy/api:${{ github.sha }} docker push sports-academy/api:latest

deploy: needs: build runs-on: ubuntu-latest environment: production steps:

- name: Deploy to Kubernetes run: |

REQUIRED TECH STACK:

Typical Tech Stack for a Sports Academy Management System

1. \*\*Frontend (User Interface)\*\*

- \*\*Frameworks/Libraries:\*\* React.js, Angular, or Vue.js

- \*\*Languages:\*\* HTML5, CSS3, JavaScript/TypeScript

- \*\*UI Libraries:\*\* Material-UI, Bootstrap, Ant Design

2. \*\*Backend (Server-side)\*\*

- \*\*Languages:\*\* Node.js (JavaScript/TypeScript), Python (Django/Flask), Java (Spring Boot), PHP (Laravel), or Ruby on Rails

- \*\*Frameworks:\*\* Express.js (for Node.js), Django/Flask (for Python), etc.

- \*\*API:\*\* RESTful API or GraphQL

3. \*\*Database\*\*

- \*\*Relational:\*\* MySQL, PostgreSQL, SQL Server

- \*\*NoSQL:\*\* MongoDB, Firebase

4. \*\*Authentication & Authorization\*\*

- \*\*Libraries/Services:\*\* JWT, OAuth, Auth0, Firebase Auth

5. \*\*Hosting/Deployment\*\*

- \*\*Cloud Providers:\*\* AWS, Azure, Google Cloud, DigitalOcean

- \*\*Platforms:\*\* Vercel, Netlify (for frontend), Heroku, Render

6. \*\*Other Tools\*\*

- \*\*Version Control:\*\* Git, GitHub/GitLab/Bitbucket

- \*\*CI/CD:\*\* GitHub Actions, GitLab CI, Jenkins

- \*\*Testing:\*\* Jest, Mocha, Cypress, Selenium

7. \*\*Optional\*\*

- \*\*Mobile App:\*\* React Native, Flutter, or native Android/iOS

- \*\*Notifications:\*\* Firebase Cloud Messaging, Twilio

- \*\*Payment Integration:\*\* Stripe, PayPal