CS-465 DISTRIBUTED DATABASE SYSTEMS

A FINAL REPORT ON THE PROJECT ENTITLED

Design and Development of a NITK Sports Facility Booking System Using Microservices Architecture

Group Members:

Shashank Prabhakar (221CS246)
Mohammad Faisal Raza (221CS137)

VII SEMESTER B-TECH CSE
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
NATIONAL INSTITUTE OF TECHNOLOGY KARNATAKA SURATHKAL
2024 – 2025

Contents

1 Abstract

This report presents the design, implementation, and evaluation of the NITK Sports Facility Booking System, a comprehensive web-based platform built using microservices architecture. The system addresses the longstanding challenge of manual and inefficient sports facility booking processes at NITK Surathkal by providing a unified, scalable, and user-friendly solution. The platform enables students to register, authenticate, browse available sports facilities (including football, badminton, cricket, and gym), reserve time slots, process payments, and receive real-time notifications. The backend architecture comprises five independent microservices—Authentication, Facility Management, Booking, Payment, and Notification—each developed using Spring Boot and maintaining its own PostgreSQL database to ensure data isolation and service autonomy. Communication between services is facilitated through RESTful APIs, with Spring Cloud Gateway serving as the centralized entry point and Netflix Eureka providing service discovery capabilities. The frontend is implemented as a React-based single-page application offering intuitive navigation and responsive user interactions. The system emphasizes key distributed systems principles including modularity, scalability, fault isolation, and independent deployability. Containerization using Docker and orchestration setup ensures consistent deployment across environments. Through comprehensive testing and evaluation, the system demonstrates successful implementation of all core functionalities while maintaining system reliability and preventing common issues such as double-booking. This report details the project motivation, architectural design decisions, implementation specifics, testing strategies, security considerations, and potential extensions toward production-ready deployment, thereby showcasing the practical application of microservices architecture in solving real-world campus management challenges.

2 Introduction

2.1 Background and Motivation

Sports and fitness activities form an integral part of student life at academic institutions, contributing significantly to physical well-being, stress management, and community building. At the National Institute of Technology Karnataka (NITK) Surathkal, sports facilities including football fields, badminton courts, cricket grounds, and gymnasium equipment are extensively utilized by students, faculty, and staff. However, the current system for managing and booking these facilities predominantly relies on manual processes or fragmented semi-digital solutions such as physical registers, spreadsheets, or informal coordination through messaging groups.

This traditional approach presents numerous challenges that significantly impact user experience and operational efficiency. Manual booking systems are inherently prone to human error, leading to frequent double-bookings where multiple users believe they have secured the same time slot. The lack of real-time availability information forces users to physically visit facilities or contact administrators to check slot availability, resulting in wasted time and frustration. Additionally, there is no centralized platform for viewing all available facilities, their schedules, pricing information, and booking history, making it difficult for users to plan their activities effectively. Payment tracking and receipt generation are often handled separately, adding another layer of complexity and potential

for discrepancies.

From an administrative perspective, manual systems create significant overhead in terms of record-keeping, conflict resolution, and facility utilization analysis. Without proper data collection and analytics, administrators cannot easily identify peak usage times, underutilized facilities, or patterns that could inform better resource allocation decisions. Furthermore, as the student population grows and facility usage increases, these manual systems become increasingly unsustainable and fail to scale effectively.

The motivation for this project stems from the recognition that these challenges can be effectively addressed through modern software engineering practices and architectural patterns. By developing a dedicated Sports Facility Booking System using microservices architecture, we aim to create a solution that not only resolves immediate operational issues but also provides a scalable, maintainable foundation for future enhancements and expansion to other campus services.

2.2 Problem Statement

The primary problem this project addresses is the absence of an efficient, transparent, and reliable system for managing sports facility bookings at NITK. Specifically, the current situation suffers from:

- Lack of real-time availability information
- Inability to prevent double-bookings through automated conflict detection
- Absence of centralized payment processing and receipt generation
- No automated notification system for booking confirmations and reminders
- Limited accessibility as users must be physically present or contact administrators
- Poor data management making it difficult to track usage patterns and generate reports
- Lack of scalability as manual systems cannot efficiently handle increased demand during peak periods

2.3 Objectives

This project aims to achieve the following specific objectives:

First, develop a comprehensive web-based Sports Facility Booking System that enables students and staff to register accounts, authenticate securely, browse available sports facilities with detailed information, view real-time slot availability for specific dates and times, reserve slots for desired sports activities, make secure payments for bookings, receive automated notifications for confirmations and reminders, view and manage their booking history, and cancel bookings when necessary.

Second, implement a robust microservices architecture consisting of at least five independent services (Authentication, Facility Management, Booking, Payment, and Notification), each maintaining its own database following the database-per-service pattern, communicating through well-defined RESTful API interfaces, registered with a service discovery mechanism (Netflix Eureka), and accessible through a centralized API Gateway (Spring Cloud Gateway).

Third, ensure system reliability and data consistency by implementing mechanisms to

prevent double-booking through optimistic locking or pessimistic locking strategies, maintaining transaction integrity across distributed services, providing fault isolation so that failure of one service does not cascade to others, and implementing proper error handling and logging throughout the system.

Fourth, design for scalability and performance by enabling independent scaling of individual services based on load, using efficient database queries and indexing strategies, implementing caching mechanisms where appropriate, and containerizing services using Docker for consistent deployment.

Fifth, incorporate security best practices including JWT-based authentication and authorization, secure password storage using BCrypt hashing, HTTPS communication in deployment environments, input validation and sanitization to prevent injection attacks, and rate limiting to prevent abuse.

Finally, create a user-friendly interface through React-based single-page application with intuitive navigation, responsive design for various screen sizes, clear feedback messages for user actions, and consistent visual design language across all pages.

2.4 Scope

This project focuses on implementing core booking functionality for four primary sports categories: football fields, badminton courts, cricket grounds, and gymnasium facilities. The system supports basic user roles (students and administrators), with administrators having additional privileges for facility management. The payment module implements a simplified mock payment flow to demonstrate the architecture, with provisions for future integration with actual payment gateways such as Razorpay or Stripe. Notifications are initially limited to in-app notifications, with the architecture designed to support future email and push notification capabilities. The system is designed primarily for NITK Surathkal but with a modular architecture that allows adaptation to other institutions.

3 Literature Review and Existing Systems

3.1 Traditional Booking Systems

Traditional sports facility booking systems typically fall into three categories. Manual register-based systems involve physical logbooks where users write their names and desired time slots, require constant monitoring by facility staff, are prone to illegible handwriting and torn pages, and provide no mechanism for advance booking or conflict detection. Spreadsheet-based systems use shared Excel or Google Sheets documents, offer slightly better organization than physical registers, but still suffer from concurrent editing conflicts, lack proper access control, and provide limited validation of booking rules. Informal coordination systems rely on messaging groups (WhatsApp, Telegram) for announcements and bookings, create information overload with important messages getting lost, have no systematic record-keeping, and make it difficult to verify booking authenticity.

3.2 Commercial Sports Booking Platforms

Several commercial platforms have successfully addressed sports facility booking challenges at scale. Playo is a comprehensive sports facility discovery and booking platform covering multiple cities in India, offering features such as venue discovery, instant booking, payment integration, community features, and partner management. Decathlon provides in-store sports activity bookings, integrates with retail operations, handles equipment rental alongside court bookings, and focuses on user convenience. PlaySpots is an Indian startup focused on urban sports facilities, emphasizes real-time availability, provides mobile-first experience, and integrates coaching services. Hudle offers corporate and community sports facility management, provides advanced analytics for facility owners, supports membership management, and includes tournament organization features.

These platforms demonstrate the viability and value of digital booking systems but are typically designed for commercial contexts with different requirements than academic institutions. They often include features unnecessary for campus use (such as marketplace dynamics or advertising) while lacking specific academic features (such as integration with campus ID systems or free facility access for students).

3.3 Microservices Architecture

Microservices architecture has emerged as a powerful pattern for building scalable, maintainable distributed systems. The core principles include service independence where each microservice focuses on a specific business capability, decentralized data management with each service owning its database, independent deployability allowing services to be updated without affecting others, technology heterogeneity enabling different services to use different technology stacks if appropriate, and resilience through isolation where failure in one service does not necessarily cascade to others.

Key architectural patterns relevant to this project include the API Gateway pattern that provides a single entry point for clients, handles routing to appropriate services, implements cross-cutting concerns (authentication, logging, rate limiting), and simplifies client interactions by aggregating multiple service calls. Service Discovery pattern enables services to find each other dynamically, supports dynamic scaling by automatically registering new instances, eliminates need for hardcoded service locations, and commonly implemented using tools like Netflix Eureka, Consul, or etcd. Database per Service pattern ensures each microservice has its own database, prevents tight coupling through shared data structures, allows each service to choose optimal database technology, but requires careful design of inter-service communication for data consistency. The Circuit Breaker pattern prevents cascading failures, automatically detects failing services, temporarily blocks requests to failing services, and allows gradual recovery.

3.4 Related Academic Projects

Several academic projects have explored similar domains. Event management systems have been implemented using microservices for handling registrations, ticketing, and notifications, demonstrating successful separation of concerns and scalable architectures. Campus resource booking systems for classrooms and laboratories have shown the value of centralized platforms with role-based access control. Healthcare appointment systems provide insights into time-slot management, conflict prevention, and notification handling

in high-stakes environments. E-commerce platforms demonstrate payment integration, transaction management, and inventory control applicable to facility booking.

3.5 Gap Analysis and Motivation

While commercial platforms demonstrate the value of digital booking systems and academic projects provide implementation insights, there remains a gap in open-source, institution-specific solutions that can be customized to unique campus requirements, integrate with existing campus infrastructure, provide transparent cost structure without subscription fees, allow complete control over data privacy and security, and serve as educational tools for understanding distributed systems.

This project aims to fill this gap by creating a comprehensive, well-documented sports facility booking system specifically designed for academic institutions like NITK, while simultaneously serving as a practical learning platform for distributed database systems and microservices architecture.

4 System Requirements

4.1 Functional Requirements

4.1.1 User Management

- FR1: The system shall allow new users to register by providing their name, email address, and password
- FR2: The system shall validate that email addresses are in proper format and are unique
- FR3: The system shall securely hash and store user passwords using BCrypt algorithm
- FR4: The system shall allow registered users to log in using their email and password
- FR5: The system shall generate and return JWT tokens upon successful authentication
- FR6: The system shall allow users to view and update their profile information
- FR7: The system shall support basic role-based access (student and administrator roles)

4.1.2 Facility Management

- FR8: The system shall maintain a catalog of sports facilities including football fields, badminton courts, cricket grounds, and gymnasium
- FR9: Each facility shall have attributes including name, description, location, capacity, and pricing information
- FR10: The system shall allow administrators to add new facilities
- FR11: The system shall allow administrators to update facility information
- FR12: The system shall allow administrators to delete or deactivate facilities
- FR13: The system shall define available time slots for each facility
- FR14: The system shall allow administrators to configure slot duration, availability windows, and pricing

- FR15: The system shall display facility availability in real-time based on existing bookings

4.1.3 Booking Management

- FR16: The system shall allow authenticated users to view all available facilities
- FR17: The system shall allow users to check slot availability for a specific facility and date
- FR18: The system shall enable users to select and reserve available time slots
- FR19: The system shall prevent double-booking by ensuring a slot can only be reserved by one user
- FR20: The system shall calculate total booking cost based on facility pricing and slot duration
- FR21: The system shall create a pending booking record upon slot selection
- FR22: The system shall allow users to view their booking history
- FR23: The system shall allow users to cancel bookings subject to cancellation policy
- FR24: The system shall automatically update facility availability when bookings are created or cancelled

4.1.4 Payment Processing

- FR25: The system shall support payment processing for confirmed bookings
- FR26: The system shall implement a mock payment flow for demonstration purposes
- FR27: The system shall record payment transactions with details including amount, timestamp, and status
- FR28: The system shall link each payment to its corresponding booking
- FR29: The system shall allow users to view their payment history
- FR30: The system shall support future integration with external payment gateways
- FR31: The system shall generate payment receipts with transaction details

4.1.5 Notification Management

- FR32: The system shall send notifications to users upon successful booking confirmation
- FR33: The system shall send reminder notifications before scheduled activities
- FR34: The system shall send notifications when bookings are cancelled
- FR35: The system shall initially support in-app notifications
- FR36: The system shall allow users to view all their notifications
- FR37: The system shall allow users to mark notifications as read
- FR38: The system shall display unread notification count to users
- FR39: The system shall support future extension to email and push notifications

4.2 Non-Functional Requirements

4.2.1 Performance Requirements

- NFR1: The system shall respond to user queries within 2 seconds under normal load
- NFR2: The system shall support at least 100 concurrent users without degradation

- NFR3: The system shall handle peak booking periods efficiently
- NFR4: Database queries shall be optimized with appropriate indexing strategies

4.2.2 Scalability Requirements

- NFR5: Individual microservices shall be independently scalable based on load
- NFR6: The system architecture shall support horizontal scaling by adding service instances
- NFR7: The database design shall accommodate growth in users, facilities, and bookings
- NFR8: The system shall be containerized to enable easy deployment and scaling

4.2.3 Reliability and Availability

- NFR9: The system shall aim for 99% uptime during operational hours
- NFR10: Failure of one microservice shall not cause complete system failure
- NFR11: The system shall implement proper error handling and graceful degradation
- NFR12: Critical data shall be backed up regularly
- NFR13: The system shall recover automatically from transient failures

4.2.4 Security Requirements

- NFR14: User passwords shall be hashed using BCrypt with appropriate salt rounds
- NFR15: Authentication shall use JWT tokens with reasonable expiration times
- NFR16: API endpoints shall validate and sanitize all user inputs
- NFR17: The system shall implement HTTPS for all communications in production
- NFR18: Sensitive configuration shall be externalized
- NFR19: The system shall implement rate limiting to prevent abuse
- NFR20: The system shall protect against common vulnerabilities

5 Proposed System Architecture

5.1 High-Level Architecture Overview

The NITK Sports Facility Booking System follows a distributed microservices architecture designed to maximize modularity, scalability, and maintainability. The system is composed of several independent services that communicate through well-defined interfaces while maintaining their own data stores and business logic.

The architecture consists of the following major components:

Client Layer: A React-based single-page application (SPA) that provides the user interface. The frontend communicates exclusively with the API Gateway and never directly with individual microservices.

API Gateway: Implemented using Spring Cloud Gateway, serves as the single entry point for all client requests. The gateway handles request routing, authentication enforcement, cross-cutting concerns such as logging and rate limiting, and protocol translation if needed.

Service Discovery: Netflix Eureka Server enables dynamic service registration and discovery. Microservices register themselves with Eureka upon startup, and the API Gateway uses Eureka to discover and route requests to available service instances.

Microservices Layer: Five independent services that encapsulate specific business capabilities: Authentication Service for user management and authentication, Facility Service for sports facility and slot management, Booking Service for reservation handling, Payment Service for transaction processing, and Notification Service for user communications.

Data Layer: Each microservice maintains its own PostgreSQL database instance, ensuring data isolation and service autonomy. This follows the database-per-service pattern fundamental to microservices architecture.

5.2 Microservices Components

5.2.1 Authentication Service

The Authentication Service is responsible for all user-related operations including registration, login, profile management, and JWT token generation and validation.

Core Responsibilities:

- User registration with email uniqueness validation
- Password hashing using BCrypt
- User authentication against stored credentials
- JWT token generation with appropriate claims
- Token validation and refresh mechanisms
- User profile retrieval and updates

Key Endpoints:

- POST /api/auth/register – User registration
- POST /api/auth/login – Authentication and token generation
- GET /api/auth/profile – Retrieve logged-in user details
- PUT /api/auth/profile – Update user information
- POST /api/auth/refresh – Refresh expired tokens

Database Schema: The auth_db database contains a users table with columns: id (primary key), name, email (unique), password_hash, role, created_at, and updated_at.

5.2.2 Facility Service

The Facility Service manages all information related to sports facilities, their available time slots, and real-time availability status.

Core Responsibilities:

- Maintaining catalog of sports facilities
- Defining time slot templates for each facility
- Calculating slot availability based on bookings
- Providing facility search and filtering capabilities
- Managing facility-related metadata

Key Endpoints:

- GET /api/facilities – List all facilities
- GET /api/facilities/{id} – Detailed facility information
- GET /api/facilities/{id}/availability?date=YYYY-MM-DD – Check available slots
- POST /api/facilities – Create new facilities (admin)
- PUT /api/facilities/{id} – Update facility information (admin)
- DELETE /api/facilities/{id} – Remove facilities (admin)

5.2.3 Booking Service

The Booking Service handles all slot reservation logic, including conflict detection, booking creation, cancellation, and status management.

Core Responsibilities:

- Validating booking requests against availability
- Preventing double-booking through concurrency control
- Creating and managing booking records
- Calculating booking costs
- Handling booking cancellations and refunds
- Maintaining booking history for users

Concurrency Control: The service implements optimistic locking using a version field or timestamp to prevent double-booking. When creating a booking, the service queries available slots, attempts to insert the booking, and handles unique constraint violations gracefully.

5.2.4 Payment Service

The Payment Service manages payment transactions, initially through a mock implementation with provisions for future gateway integration.

Core Responsibilities:

- Recording payment transactions
- Linking payments to bookings
- Generating transaction IDs
- Storing payment status and timestamps
- Providing payment history to users
- Supporting future integration with external payment gateways

5.2.5 Notification Service

The Notification Service handles all user notifications, initially supporting in-app notifications with extensibility for email and push notifications.

Core Responsibilities:

- Creating notification records for booking events
- Storing notification content and metadata
- Marking notifications as read/unread
- Retrieving user notification history

- Supporting future multi-channel notification delivery

5.3 Inter-Service Communication

Services in this architecture communicate primarily through synchronous RESTful APIs. The Booking Service calls Facility Service to verify slot availability before creating bookings. After successful booking creation, Booking Service calls Payment Service to initiate payment. Upon payment confirmation, Payment Service can trigger Notification Service to send confirmation messages. The API Gateway handles all external requests and routes them to appropriate services using Eureka service discovery.

5.4 Data Architecture

The system follows the database-per-service pattern. Each service has its own PostgreSQL database instance, ensuring data encapsulation and service independence. This provides benefits such as independent development and deployment, schema evolution without affecting other services, optimal data models for each service's needs, and failure isolation.

6 Implementation Details

6.1 Technology Stack

The implementation leverages a modern technology stack:

- **Backend Framework:** Spring Boot 3.x with Spring Cloud components
- **Frontend Framework:** React 18.x with React Router
- **Databases:** PostgreSQL for all microservices
- **Authentication:** JWT (JSON Web Tokens)
- **API Communication:** RESTful APIs with JSON
- **Containerization:** Docker and Docker Compose
- **Development Tools:** Maven, Git, IntelliJ IDEA, Postman

6.2 Authentication Service Implementation

User Registration Flow:

1. Client submits registration request with name, email, and password
2. Service validates email format and uniqueness
3. Password is hashed using BCrypt with 12 rounds
4. User record is created in auth_db
5. Success response is returned to client

Login Flow:

1. Client submits login credentials
2. Service retrieves user by email
3. Password is verified against stored hash
4. JWT token is generated with user claims
5. Token with expiration time is returned

6.3 Facility Service Implementation

The Facility Service manages sports facilities and their availability. Administrators can create facilities with name, type, description, location, capacity, and pricing. For each facility, available time slots are defined. When queried for a specific date, the service determines applicable slots based on day of week, queries the Booking Service to check existing bookings, calculates available time windows, and returns list of bookable slots.

6.4 Booking Service Implementation

The Booking Service implements core reservation logic with careful attention to concurrency. To prevent double-booking, the service uses database-level unique constraints on (facility_id, slot_date, slot_start_time). When creating a booking, a transaction is opened, booking insert is attempted, and if constraint violation occurs, appropriate error is returned.

Booking Status State Machine:

- PENDING: Initial state when booking is created
- CONFIRMED: After successful payment
- CANCELLED: When user cancels or payment fails
- COMPLETED: After the scheduled time has passed

6.5 Payment Service Implementation

The Payment Service handles transaction processing through mock implementation. For this prototype, payments are automatically approved. Client calls POST /api/payments with booking_id and amount. Service validates booking exists and amount matches. Unique transaction_id is generated using UUID. Payment record is created with SUCCESS status. Confirmation is returned to client.

6.6 Notification Service Implementation

The Notification Service manages user communications about booking events. Notifications are stored in notification_db and retrieved via API. Each notification has type, title and message content, read/unread status, and creation timestamp.

6.7 API Gateway Implementation

Spring Cloud Gateway routes requests and implements cross-cutting concerns. Routes are defined specifying path predicates, destination service URIs, and optional filters. The gateway enables CORS to allow frontend access and provides centralized error handling.

6.8 Frontend Implementation

The React frontend provides an intuitive single-page application experience with components for authentication, facility browsing, booking management, payment processing, and notifications. Protected routes ensure only authenticated users access certain features.

6.9 Database Design and Schema

Each service maintains its own PostgreSQL database:

Authentication Database: Users table with id, name, email, password_hash, role, and timestamps.

Facility Database: Facilities table with facility details and slots table defining available time windows.

Booking Database: Bookings table with unique constraint on facility_id, slot_date, and slot_start_time to prevent double-booking.

Payment Database: Payments table linking to bookings with transaction details.

Notification Database: Notifications table with user notifications and read status.

6.10 Containerization and Deployment

The system is containerized using Docker. Each service has a Dockerfile. A docker-compose configuration defines all services and their dependencies with environment variables for configuration and volumes for data persistence.

7 Testing and Evaluation

7.1 Testing Strategy

The testing approach combines multiple levels:

Unit Testing: Individual service methods tested in isolation using JUnit 5 framework. Mock objects (Mockito) simulate dependencies. Test coverage focused on business logic and edge cases.

Integration Testing: API endpoints tested with Postman collections. Database operations verified with test data. Inter-service communication tested through API Gateway.

End-to-End Testing: Complete user scenarios tested from frontend to database. Registration, login, browse, book, pay, notify flow validated. Admin scenarios tested. Concurrent user scenarios tested for race conditions.

7.2 Unit Testing

Each microservice includes JUnit tests for core functionality.

Authentication Service Tests:

- testUserRegistration_Success
- testUserRegistration_DuplicateEmail
- testLogin_ValidCredentials
- testLogin_InvalidPassword
- testJwtTokenGeneration
- testJwtTokenValidation

Booking Service Tests:

- testCreateBooking_AvailableSlot
- testCreateBooking_DoubleBooking
- testCancelBooking_WithinPolicy
- testCancelBooking_PastDeadline
- testCalculateCost

7.3 Integration Testing

Postman collections test API endpoints across services covering authentication, facility management, booking operations, payment processing, and notification delivery.

7.4 End-to-End Testing

Complete user workflows tested through the frontend:

Test Case 1: New User Complete Booking Flow

1. Navigate to registration page
2. Submit valid registration form
3. Login with registered credentials
4. Browse facility list, select facility
5. View facility details and available slots
6. Select slot and confirm booking
7. Complete mock payment
8. Verify booking confirmation
9. Check bookings, payments, and notifications pages

Test Case 2: Concurrent Booking Attempt

1. Open two browser windows with different users
2. Both users select same slot
3. User A books first
4. User B receives "Slot no longer available" error
5. Verify only one booking exists

7.5 Performance Testing

Basic load testing conducted using Apache JMeter with 50 concurrent virtual users over 10 minutes. Results showed average response time of 245ms, 95th percentile of 580ms, throughput of 180 requests/second, and error rate of 0.3%.

7.6 Security Testing

Security aspects verified:

- Passwords stored as BCrypt hashes
- JWT token expiration working correctly
- Invalid tokens rejected with 401 status
- Protected routes block unauthenticated requests
- SQL injection attempts rejected
- XSS attempts sanitized

7.7 Bug Fixes and Issues Resolved

Several issues were identified and fixed:

Issue 1 - Gateway 503 Errors: Resolved by adding health check delays and increasing Eureka fetch interval.

Issue 2 - Double Booking Race Condition: Fixed with unique database constraint and optimistic locking.

Issue 3 - Token Expiration Handling: Added token expiration check and graceful logout.

Issue 4 - Slow Facility Search: Added indexes on facility columns.

Issue 5 - Notification Counter Not Updating: Implemented callback to refresh count.

7.8 Evaluation Results

All five microservices implemented and functioning correctly. Frontend provides complete user experience. Database-per-service pattern successfully implemented. Service discovery and API Gateway working as designed. Booking conflict prevention working reliably. All requirements from proposal satisfied.

8 Security Considerations

8.1 Authentication and Authorization

Password Security: Passwords hashed using BCrypt with salt factor 12. Plain text passwords never stored or logged.

JWT Token Security: Tokens signed with HS256 algorithm. Secret key stored in environment variables. Token expiration set to 24 hours.

Role-Based Access Control: Users assigned roles (STUDENT, ADMIN). Admin-only endpoints protected with role checks.

8.2 Data Protection

Database Security: Each service database isolated. Database credentials stored in environment variables. Prepared statements prevent SQL injection.

Input Validation: All user inputs validated on backend. Email format validation. SQL injection prevention through ORM. XSS prevention through output encoding.

8.3 API Security

CORS Configuration: Gateway configured with specific allowed origins. Frontend URL whitelisted.

Rate Limiting: Prevents brute-force attacks. Limits requests per user/IP.

Error Handling: Error messages don't reveal sensitive information. Stack traces not exposed to clients.

9 Limitations and Challenges

9.1 Current Limitations

Payment Integration: Mock payment system not suitable for production. No real transaction processing.

Notification Channels: Only in-app notifications implemented. Email and push notifications not functional.

Mobile Experience: Limited mobile responsiveness testing. No native mobile apps.

Scalability Testing: Load testing limited to moderate concurrency. Auto-scaling not implemented.

Monitoring: Basic logging only. No centralized log aggregation. Limited metrics collection.

9.2 Technical Challenges

Distributed Transactions: Maintaining consistency across services is complex. Saga pattern implementation requires careful design. Testing distributed scenarios is challenging.

Service Communication: Synchronous REST calls create coupling. Network failures require retry logic. Timeout handling needs improvement.

Data Consistency: Eventual consistency acceptable for most operations. Critical operations need stronger guarantees. Reconciliation processes needed for data drift.

9.3 Deployment Challenges

Container Orchestration: Docker Compose suitable for development only. Production needs Kubernetes or similar. Configuration management gets complex.

Database Management: Multiple databases increase operational overhead. Schema migrations must be coordinated. Backup and restore more complex.

10 Future Work and Enhancements

10.1 Short-Term Enhancements

Payment Gateway Integration: Integrate Razorpay or Stripe for real payments. Implement secure webhook handling. Add refund processing. Support multiple payment methods. Generate proper tax invoices.

Email Notifications: Integrate SendGrid or AWS SES. Create email templates for different events. Implement async email sending. Add unsubscribe functionality. Track email delivery status.

Push Notifications: Implement Firebase Cloud Messaging. Register device tokens. Send push for booking reminders. Allow notification preferences. Handle delivery status.

Mobile Responsiveness: Optimize layouts for mobile screens. Improve touch interactions. Test on various devices. Consider progressive web app. Add mobile-specific features.

10.2 Medium-Term Enhancements

Advanced Booking Features:

- Recurring bookings (weekly/monthly)
- Group bookings with multiple users
- Waitlist functionality for fully booked slots
- Equipment rental alongside facility booking
- Calendar view of bookings

Analytics Dashboard:

- Facility utilization reports
- Peak usage times analysis
- Revenue tracking
- User behavior analytics
- Predictive availability

Social Features:

- User profiles and activity feed
- Follow friends' bookings
- Challenge friends to matches
- Community leaderboards
- Photo sharing from activities

10.3 Long-Term Enhancements

Multi-Tenant Architecture: Support multiple institutions. Tenant-specific branding. Isolated data per tenant. Centralized management. Subscription pricing model.

Tournament Organization: Create and manage tournaments. Bracket generation. Score tracking. Prize distribution. Tournament registration.

Coach and Instructor Booking: List available coaches. Book coaching sessions. Rate and review coaches. Coach earnings tracking.

AI and ML Features: Predict facility demand. Recommend booking times. Suggest facilities based on preferences. Anomaly detection for fraud. Chatbot for customer support.

10.4 Infrastructure Improvements

Kubernetes Deployment: Deploy on production Kubernetes cluster. Implement horizontal pod autoscaling. Use Helm charts for configuration. Configure ingress controllers. Set up persistent volumes.

CI/CD Pipeline: Automated testing on commits. Build and push Docker images. Deploy to staging automatically. Canary deployments to production. Rollback capabilities.

Monitoring and Logging: Implement Prometheus for metrics. Grafana dashboards for visualization. ELK stack for log aggregation. Distributed tracing with Jaeger. Alert rules for critical issues.

Message Queue: Implement RabbitMQ or Kafka. Async notification sending. Event-driven architecture. Event sourcing for audit logs. Reliable message delivery.

11 Conclusion

This project successfully demonstrates the design and implementation of a Sports Facility Booking System for NITK Surathkal using microservices architecture. The system simplifies facility booking and management for students and administrators while showcasing key distributed systems concepts.

It comprises five independent microservices—Authentication, Facility Management, Booking, Payment, and Notification—each with its own database. This design enables independent scalability, fault isolation, technology flexibility, and easier maintenance.

Key achievements include JWT-based authentication, database-per-service pattern with PostgreSQL, Eureka service discovery, Gateway-based routing, optimistic locking for booking conflict prevention, and a mock payment and in-app notification system. Testing confirmed functional reliability and effective double-booking prevention.

Limitations include the lack of real payment integration, limited notification methods, and incomplete mobile optimization. Nonetheless, the system is structured for easy enhancement and real-world deployment.

Overall, the project effectively applies distributed database and microservices principles to a practical campus problem, offering valuable hands-on learning and a strong foundation for future development.

References

1. Spring Boot Reference Documentation. VMware Tanzu, 2024. Available at: https://docs.spring.io/spring-boot/
2. Spring Cloud Documentation. VMware Tanzu, 2024. Available at: https://spring.io/projects/spring-cloud
3. Richardson, C. (2018). *Microservices Patterns: With Examples in Java*. Manning Publications.
4. Newman, S. (2021). *Building Microservices: Designing Fine-Grained Systems* (2nd Edition). O'Reilly Media.
5. React Documentation. Meta Platforms, 2024. Available at: https://react.dev/
6. PostgreSQL Documentation. PostgreSQL Global Development Group, 2024. Available at: https://www.postgresql.org/docs/
7. Docker Documentation. Docker Inc., 2024. Available at: https://docs.docker.com/

8. Netflix Eureka Documentation. Netflix, 2024. Available at: https://github.com/Netflix/eureka/wiki
9. JWT.io. Auth0, 2024. JSON Web Token introduction and libraries. Available at: https://jwt.io/
10. Kleppmann, M. (2017). *Designing Data-Intensive Applications*. O'Reilly Media.

A API Endpoints Documentation

This appendix provides comprehensive documentation of all API endpoints exposed by the microservices.

A.1 Authentication Service Endpoints

Method	Endpoint	Description
POST	/api/auth/register	Register new user
POST	/api/auth/login	Authenticate and return JWT
GET	/api/auth/profile	Get user profile (auth required)
PUT	/api/auth/profile	Update user information
POST	/api/auth/refresh	Refresh JWT token

A.2 Facility Service Endpoints

Method	Endpoint	Description
GET	/api/facilities	List all facilities
GET	/api/facilities?type=BADMINTON	Filter by type
GET	/api/facilities/{id}	Get facility details
GET	/api/facilities/{id}/availability	Get available slots
POST	/api/facilities	Create facility (admin)
PUT	/api/facilities/{id}	Update facility (admin)
DELETE	/api/facilities/{id}	Delete facility (admin)

A.3 Booking Service Endpoints

Method	Endpoint	Description
POST	/api/bookings	Create new booking
GET	/api/bookings	Get user's bookings
GET	/api/bookings/{id}	Get booking details
DELETE	/api/bookings/{id}	Cancel booking
GET	/api/bookings/facility/{id}	Facility bookings (admin)

A.4 Payment Service Endpoints

Method	Endpoint	Description

POST	/api/payments	Create payment record
GET	/api/payments	Get payment history
GET	/api/payments/{id}	Get payment details
GET	/api/payments/booking/{id}	Get booking payment
POST	/api/payments/confirm	Confirm mock payment

A.5 Notification Service Endpoints

Method	Endpoint	Description
POST	/api/notifications	Create notification
GET	/api/notifications	Get user notifications
GET	/api/notifications/unread/count	Get unread count
PUT	/api/notifications/{id}/read	Mark as read
DELETE	/api/notifications/{id}	Delete notification

B Database Schema Diagrams

B.1 Authentication Database Schema

```
          users

id              SERIAL PK
name            VARCHAR(255)
email           VARCHAR(255) UK
password_hash   VARCHAR(255)
role            VARCHAR(50)
created_at      TIMESTAMP
updated_at      TIMESTAMP
```

B.2 Facility Database Schema

```
         facilities

id              SERIAL PK
name            VARCHAR(255)
type            VARCHAR(50)
description     TEXT
location        VARCHAR(255)
capacity        INTEGER
price_per_hour  DECIMAL(10,2)
status          VARCHAR(50)
created_at      TIMESTAMP
```

```
          1:N


            slots

 id               SERIAL PK
 facility_id      INTEGER FK
 day_of_week      INTEGER
 start_time       TIME
 end_time         TIME
 is_active        BOOLEAN
```

B.3 Booking Database Schema

```
           bookings

 id                  SERIAL PK
 user_id             INTEGER
 facility_id         INTEGER
 slot_date           DATE
 slot_start_time     TIME
 slot_end_time       TIME
 duration_hours      DECIMAL(4,2)
 total_cost          DECIMAL(10,2)
 status              VARCHAR(50)
 booking_time        TIMESTAMP
 cancellation_time   TIMESTAMP

UNIQUE INDEX on (facility_id, slot_date,
  slot_start_time) WHERE status != 'CANCELLED'
```

B.4 Payment Database Schema

```
           payments

 id                 SERIAL PK
 booking_id         INTEGER UK
 user_id            INTEGER
 amount             DECIMAL(10,2)
 currency           VARCHAR(10)
 transaction_id     VARCHAR(255)UK
 payment_method     VARCHAR(50)
 status             VARCHAR(50)
 created_at         TIMESTAMP
```

```
updated_at        TIMESTAMP
```

B.5 Notification Database Schema

```
        notifications

id                SERIAL PK
user_id           INTEGER
type              VARCHAR(50)
title             VARCHAR(255)
message           TEXT
is_read           BOOLEAN
created_at        TIMESTAMP
read_at           TIMESTAMP
```

C Deployment Guide

C.1 Local Development Setup

Prerequisites:

- Java 17 or higher
- Maven 3.8+
- Node.js 18+ and npm
- Docker and Docker Compose
- PostgreSQL (if running without Docker)

Steps to run with Docker:

```
# Clone repository
git clone <repository-url>
cd nitk-sports-booking

# Create .env file
echo "JWT_SECRET=your-secret-key" > .env

# Build all services
./build-all.sh

# Start all services
docker-compose up -d

# View logs
docker-compose logs -f

# Access application
# Frontend: http://localhost:3000
# API Gateway: http://localhost:8080
```

```
# Eureka: http://localhost:8761
```

D Individual Contributions

D.1 Shashank Prabhakar (221CS246)

Primary Responsibilities:

- Backend architecture design
- Authentication Service implementation
- Booking Service with concurrency control
- API Gateway configuration
- Report contribution
- Frontend development with React
- UI/UX design
- Frontend-backend integration

D.2 Mohammad Faisal Raza (221CS137)

Primary Responsibilities:

- Facility Service development
- Payment Service implementation
- Notification Service creation
- Eureka Server setup
- Database schema design
- Integration testing with Postman
- Docker containerization
- End-to-end testing

D.3 Collaborative Efforts

Both team members collaborated on:

- Project planning and architecture
- Database schema design
- API contract definition
- Testing strategy
- Troubleshooting and debugging
- Documentation and report
- Presentation preparation

Submitted by:
Shashank Prabhakar (221CS246)
Mohammad Faisal Raza (221CS137)