Smart Campus Navigation and Facility Booking System

COSC333 Software Systems & Design

Technical Implementation & Live Demo

Professor: Dr. Kamal Taha | **By:** Salem Alhaddad | **Date:** June 26, 2025

🔧 Node.js + Express.js | 🗄️ PostgreSQL | 🔐 JWT Authentication

🎯 Requirements Engineering & Problem Analysis

- **Functional Requirements:** Interactive navigation, real-time booking, role-based access control
- **Non-Functional Requirements:** Performance (<2s response), security (JWT), scalability (100+ users)
- **Stakeholder Analysis:** Students, Faculty, Staff, Administrators, Visitors with distinct needs
- **Use Case Modeling:** 19 identified use cases with actor-system interactions
- **Requirements Traceability:** Each requirement mapped to implementation and testing
- **Risk Assessment:** Booking conflicts, security vulnerabilities, performance bottlenecks

🏗️ Software Architecture & Design Patterns

- **Architectural Pattern:** Layered Architecture (Presentation → Business → Data)
- **Design Patterns:** MVC separation, Repository pattern, Middleware pattern
- **SOLID Principles:** Single responsibility, Open/closed, Dependency inversion
- **Modular Design:** Separation of concerns with distinct service layers
- **API Design:** RESTful architecture following Richardson Maturity Model
- **Database Design:** Entity-Relationship modeling with normalization

```
// Layered Architecture Implementation
// Presentation Layer → Business Logic → Data Access Layer
const app = express();
app.use(helmet()); // Security middleware
app.use('/api/auth', authRoutes); // Authentication layer
app.use('/api/bookings', authenticate, bookingRoutes); // Business logic
```

📊 UML Modeling & Database Design

- **UML Diagrams:** Use Case, Class, Sequence, Component diagrams following UML 2.5
- **Entity-Relationship Design:** 8 entities with proper normalization (3NF)
- **Class Modeling:** Domain objects with attributes, methods, and associations
- **Sequence Diagrams:** Authentication, booking, navigation interaction flows
- **Database Schema:** PostgreSQL with ACID transactions and referential integrity
- **Data Modeling:** Conceptual → Logical → Physical design progression

```sql
-- Entity-Relationship Implementation with Constraints
CREATE TABLE bookings (
  booking_id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
  room_id UUID REFERENCES rooms(room_id) ON DELETE CASCADE,
  user_id UUID REFERENCES users(user_id) ON DELETE CASCADE,
  start_time TIMESTAMP NOT NULL,
  end_time TIMESTAMP NOT NULL,
  CHECK (end_time > start_time) -- Business rule constraint
);
```

🎮 LIVE DEMO

Demo URL: http://localhost:3000

🔐 Security Architecture

- **Password Hashing:** bcrypt with 12 salt rounds
- **JWT Tokens:** 24h expiration + refresh token rotation
- **Input Validation:** express-validator with custom rules
- **SQL Injection Prevention:** Parameterized queries only
- **Rate Limiting:** 100 requests/15min per IP
- **Security Headers:** Content Security Policy + HSTS

```
// JWT Authentication Middleware
const authenticate = async (req, res, next) => {
  const token = req.header('Authorization')?.replace('Bearer ', '');
  try {
    const decoded = jwt.verify(token, process.env.JWT_SECRET);
    const user = await User.findById(decoded.userId);
    if (!user?.isActive) throw new Error('User deactivated');
    req.user = user;
    next();
  } catch (error) {
    res.status(401).json({ success: false, error: 'Invalid token' });
  }
};
```

🔌 Software Testing & Quality Assurance

- **Testing Strategy:** Test pyramid with Unit (70%), Integration (20%), E2E (10%)
- **Test-Driven Development:** Red-Green-Refactor cycle for core functions
- **Code Coverage:** 85% line coverage, 75% branch coverage, 90% function coverage
- **Quality Metrics:** Cyclomatic complexity, maintainability index, technical debt
- **API Testing:** Postman collections, automated endpoint validation
- **Performance Testing:** Load testing with 100 concurrent users, <2s response time

```
// Unit Testing Example - TDD Approach
describe('Room Availability Service', () => {
  test('should return false for overlapping bookings', async () => {
    // Arrange
    const room = await createTestRoom();
    await createBooking(room.id, '09:00', '10:00');

    // Act
    const isAvailable = await checkAvailability(room.id, '09:30', '10:30');

    // Assert
    expect(isAvailable).toBe(false);
  });
});
```

📁 Software Development Process & Methodology

- **Development Methodology:** Iterative and Incremental development approach
- **Version Control:** Git with feature branches and pull request workflow
- **Code Organization:** Package-by-feature structure with dependency injection
- **Documentation:** Technical specs, API docs, UML diagrams, user manuals
- **Configuration Management:** Environment-based deployment configurations
- **DevOps Practices:** Automated testing, continuous integration readiness

```
Development Process Applied:
├── 1. Requirements Analysis     # Stakeholder interviews, use case modeling
├── 2. System Design            # Architecture, UML diagrams, DB schema
├── 3. Implementation           # Iterative coding with testing
├── 4. Testing & Validation     # Unit, integration, system testing
├── 5. Documentation           # Technical and user documentation
└── 6. Deployment & Maintenance # Production readiness assessment
```

⚡ Performance & Scalability

- **Database Optimization:** 12 composite indexes + query optimization
- **Connection Pooling:** 20 max connections with 5s timeout
- **Frontend Performance:** <3s initial load, lazy loading components
- **Memory Management:** Efficient garbage collection + resource cleanup
- **Concurrent Users:** 100+ simultaneous sessions tested
- **API Response Time:** 95th percentile <500ms

```
// Database Connection Pool Configuration
const pool = new Pool({
  connectionString: process.env.DATABASE_URL,
  max: 20,                    // Maximum connections
  idleTimeoutMillis: 5000,    // Close idle connections
  connectionTimeoutMillis: 2000,
  ssl: { rejectUnauthorized: false }
});
```

🚀 Production Deployment

- **Environment Configuration:** Separate dev/staging/prod settings
- **Security Hardening:** HTTPS, security headers, rate limiting
- **Monitoring & Logging:** Winston + structured JSON logging
- **Error Handling:** Graceful degradation + user-friendly messages
- **Backup Strategy:** Automated database backups + point-in-time recovery
- **Scalability:** Stateless design supporting horizontal scaling

```
# Production Environment Variables
NODE_ENV=production
PORT=443
DATABASE_URL=postgresql://prod_user:secure_pass@db.host:5432/prod_db
JWT_SECRET=256-bit-production-secret
REDIS_URL=redis://redis.host:6379
RATE_LIMIT_REQUESTS=1000
```

🏆 COSC333 Learning Outcomes Achieved

- **Requirements Engineering:** Functional/non-functional requirements, stakeholder analysis
- **Software Design:** UML modeling, architectural patterns, design principles (SOLID)
- **Implementation:** Object-oriented programming, design patterns, code organization
- **Testing & QA:** Test-driven development, testing strategies, quality metrics
- **Project Management:** Iterative development, documentation, version control
- **Professional Practice:** Security, performance, maintainability, scalability

Software Engineering Metrics:

✅ 19 Use Cases documented and implemented
✅ 4 UML diagram types (Use Case, Class, Sequence, Component)
✅ 8 Entity relationships with proper normalization
✅ 85% test coverage demonstrating TDD practices
✅ Layered architecture with separation of concerns
✅ WCAG 2.1 AA accessibility and security best practices

❓ Questions & Course Concept Discussion

- **Requirements Engineering:** How did stakeholder analysis influence design decisions?
- **UML Modeling:** Which diagram types were most valuable for system understanding?
- **Design Patterns:** How do SOLID principles manifest in the codebase?
- **Testing Methodologies:** What TDD practices improved code quality?
- **Software Architecture:** How does layered architecture support maintainability?
- **Project Management:** What iterative development challenges were encountered?

Thank You

Questions & Live Demo

Ready for technical deep-dive discussion