EventBook System Documentation

Advanced Web Development Project

A Comprehensive Event Booking Platform

EVENT BOOKINGS

By NDE HURICH DILAN

ICTU20223351

May 2025

Covering:

- Project Overview & Architecture
- System Design & Database Schema
- Implementation Details
- Code Documentation
- Deployment Strategy

Contents

1 Project Overview **3**
1.1 Introduction . . . 3
1.2 Project Objectives . . . 3
1.3 Key Features . . . 3
1.4 Target Audience . . . 3

2 System Design **4**
2.1 Architecture Overview . . . 4
2.2 Design Patterns Used . . . 4
2.3 Database Design . . . 4
2.4 Class Diagram . . . 5
2.5 ER Diagram . . . 6
2.6 Use Case Diagram . . . 7
2.7 Sequence Diagrams . . . 8
2.7.1 Event Booking Sequence . . . 8
2.7.2 User Authentication Sequence . . . 9

3 Implementation **10**
3.1 Technology Stack . . . 10
3.2 Project Structure . . . 10
3.3 Architecture Rationale . . . 11
3.3.1 Separation of Concerns . . . 11
3.3.2 Scalability Considerations . . . 11
3.4 Key Implementation Decisions . . . 11
3.4.1 Database Connection Management . . . 11
3.4.2 Frontend JavaScript Architecture . . . 12
3.5 Database Schema Implementation . . . 13
3.6 Security Implementation . . . 14
3.6.1 Authentication System . . . 14
3.6.2 Data Protection . . . 14

4 Deployment Strategy **14**
4.1 Cloud Deployment Options . . . 14
4.1.1 Recommended Platform: DigitalOcean . . . 14
4.1.2 Alternative Platforms . . . 15
4.2 Deployment Architecture . . . 15
4.3 Performance Optimization . . . 15

5 Code Documentation **15**
5.1 API Endpoints . . . 15
5.2 Event Booking Process Flow . . . 15
5.3 Frontend-Backend Communication . . . 16
5.4 Security Measures in Code . . . 17
5.4.1 Input Sanitization . . . 17
5.4.2 Authentication Check . . . 17
5.5 Performance Optimization Code . . . 18

5.5.1 Database Query Optimization 18

6 Future Enhancements 19

6.1 Planned Features 19

6.2 Scalability Considerations 19

7 Conclusion 19

1 Project Overview

1.1 Introduction

EventBook is a comprehensive web-based event booking system designed specifically for the Cameroonian market. The platform enables users to discover, book, and manage event tickets while providing administrators with powerful tools for event management, booking analytics, and customer relationship management.

1.2 Project Objectives

The primary objectives of this project include:

- **User-Friendly Event Discovery**: Provide an intuitive interface for users to browse and search events
- **Seamless Booking Process**: Implement a streamlined cart-to-checkout flow
- **Administrative Control**: Offer comprehensive admin tools for event and booking management
- **Cultural Relevance**: Focus on Cameroonian events and cultural celebrations
- **Scalable Architecture**: Build a system that can accommodate growth and additional features

1.3 Key Features

Core Features

- **Event Management**: Create, edit, and manage events with detailed information
- **User Authentication**: Secure registration and login system
- **Shopping Cart**: Add multiple events to cart before checkout
- **Booking System**: Complete booking process with reference generation
- **Admin Dashboard**: Comprehensive administrative interface
- **Responsive Design**: Mobile-friendly interface using Bootstrap
- **Search & Filter**: Advanced event discovery capabilities
- **Ticket Management**: Digital ticket generation and download

1.4 Target Audience

- **Primary Users**: Event attendees seeking cultural, business, and entertainment events in Cameroon
- **Event Organizers**: Organizations and individuals hosting events
- **System Administrators**: Platform managers overseeing operations

2 System Design

2.1 Architecture Overview

The EventBook system follows a three-tier architecture pattern:

1. **Presentation Layer**: HTML, CSS, JavaScript (Bootstrap framework)
2. **Application Layer**: PHP backend with RESTful API endpoints
3. **Data Layer**: MySQL database with normalized schema

2.2 Design Patterns Used

- **MVC Pattern**: Separation of concerns between models, views, and controllers
- **Repository Pattern**: Database abstraction for data access
- **Singleton Pattern**: Database connection management
- **RESTful API Design**: Stateless API endpoints for frontend-backend communication

2.3 Database Design

The system uses a normalized MySQL database with the following core entities:

Entity	**Purpose**	**Key Relationships**
Users	User account management	One-to-many with Bookings, Cart
Events	Event information storage	One-to-many with Booking Items
Bookings	Order management	Many-to-one with Users
Booking Items	Individual event bookings	Many-to-one with Events, Bookings
Cart	Shopping cart functionality	Many-to-one with Users, Events

Table 1: Core Database Entities

2.4 Class Diagram

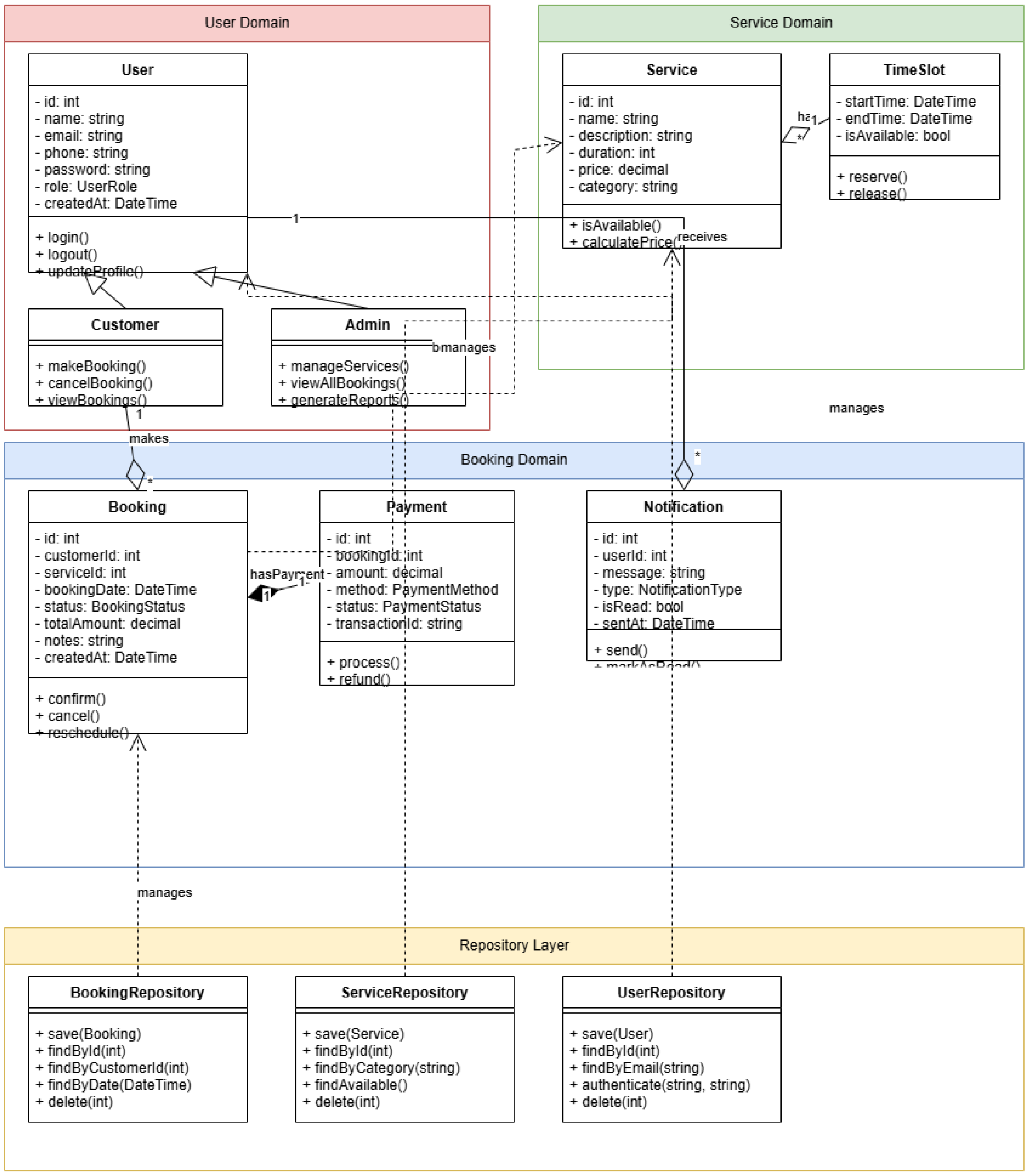

Figure 1: System Class Diagram showing main entities and relationships

2.5 ER Diagram

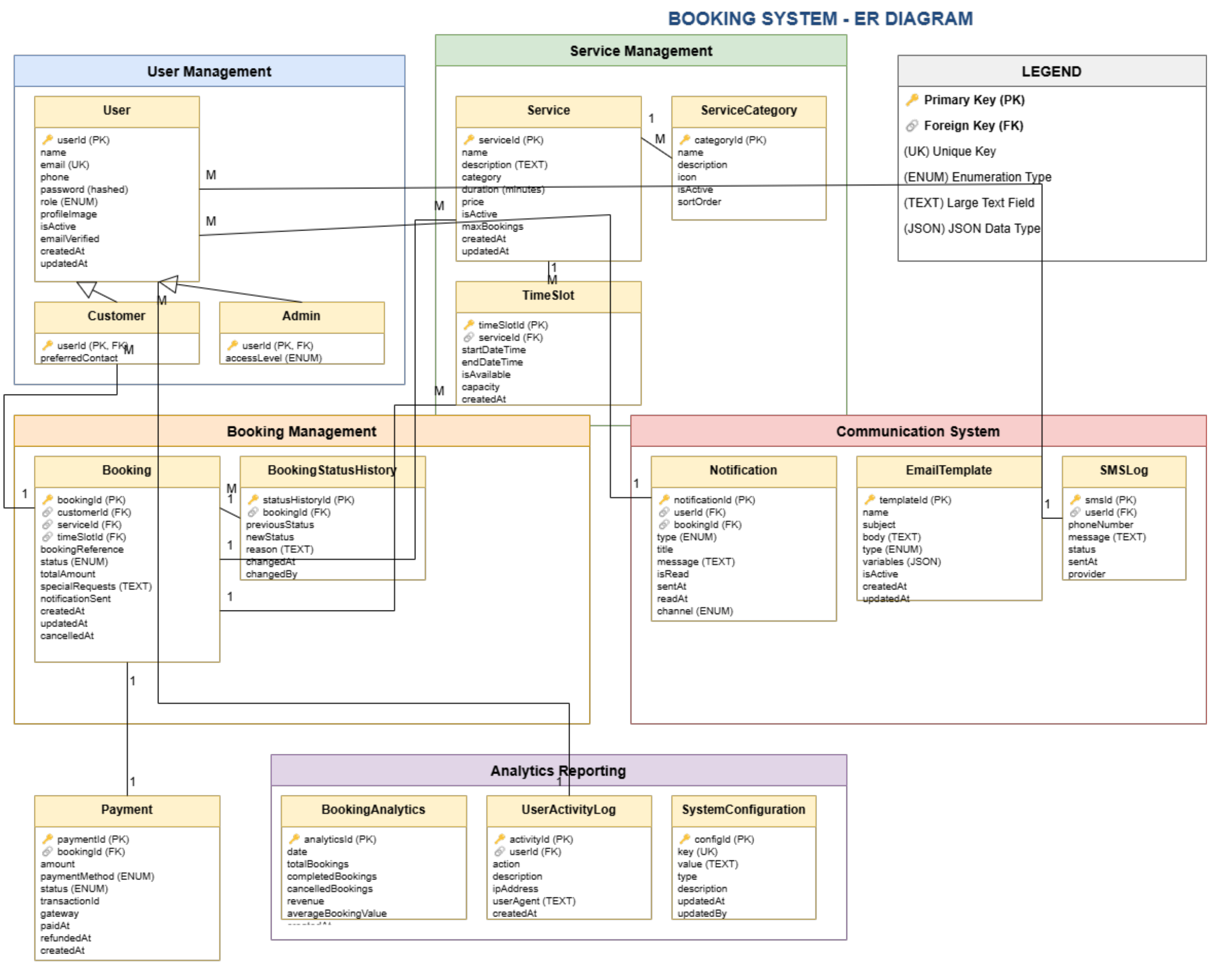

Figure 2: ER Diagram showing the entities of the system

2.6 Use Case Diagram

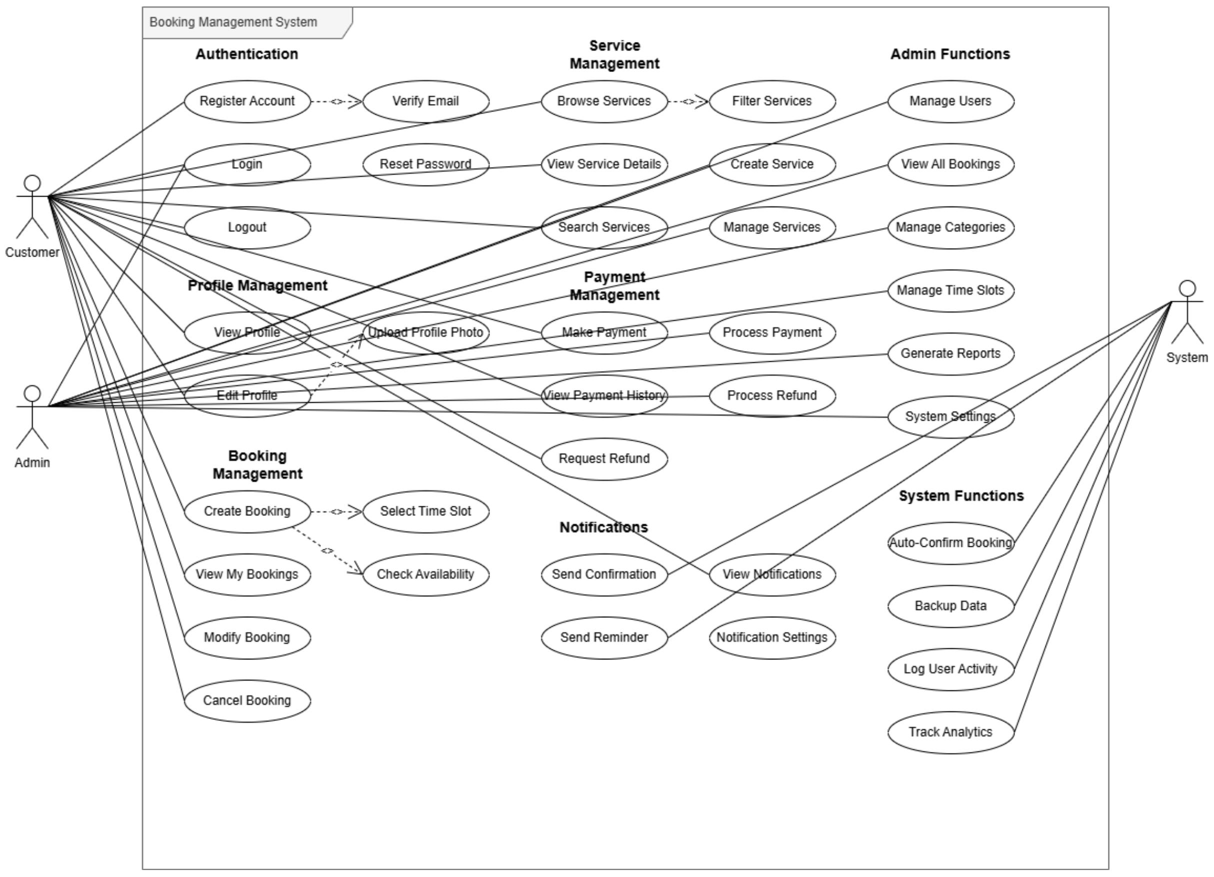

Figure 3: Use Case Diagram showing user interactions with the system

2.7 Sequence Diagrams

2.7.1 Event Booking Sequence

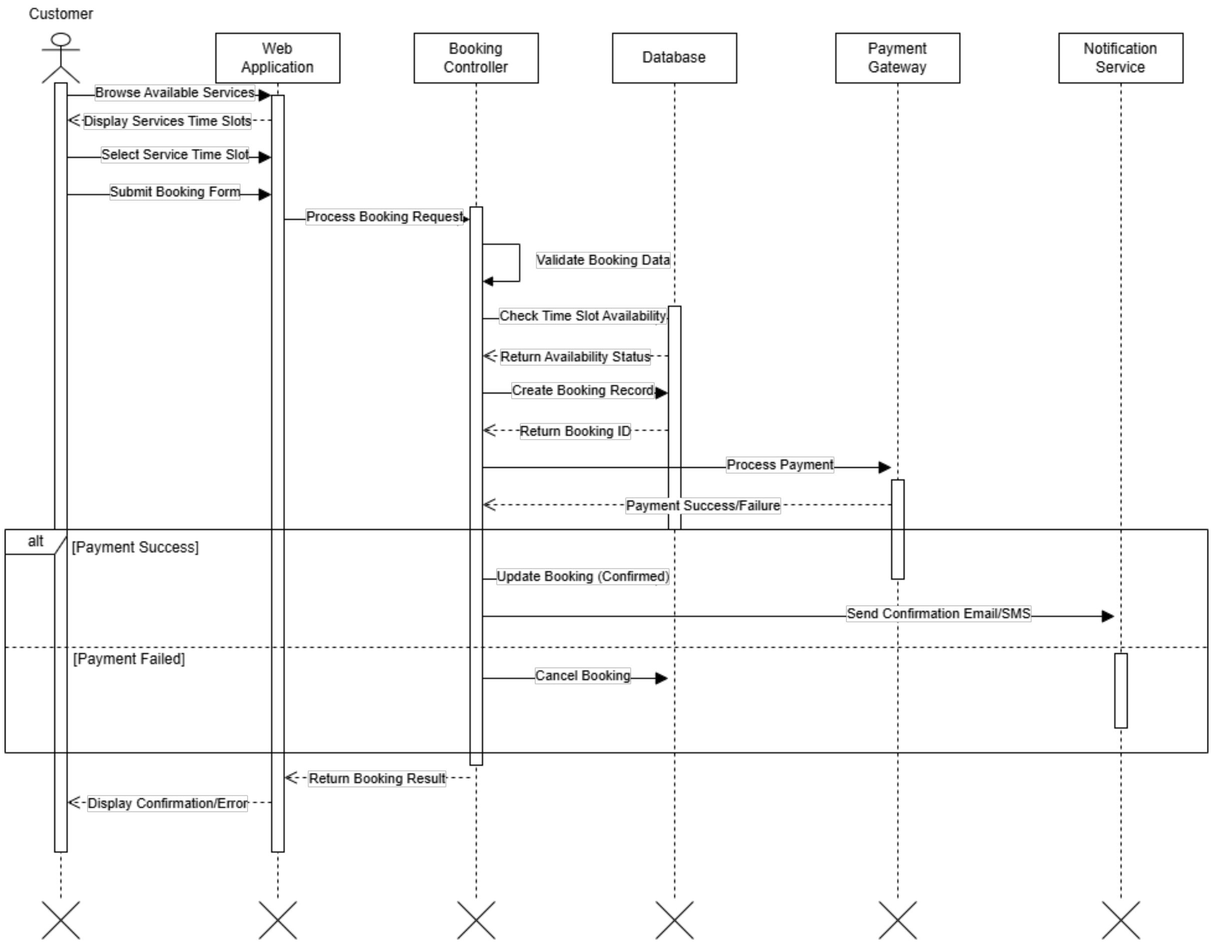

Figure 4: Sequence diagram for the event booking process

2.7.2 User Authentication Sequence

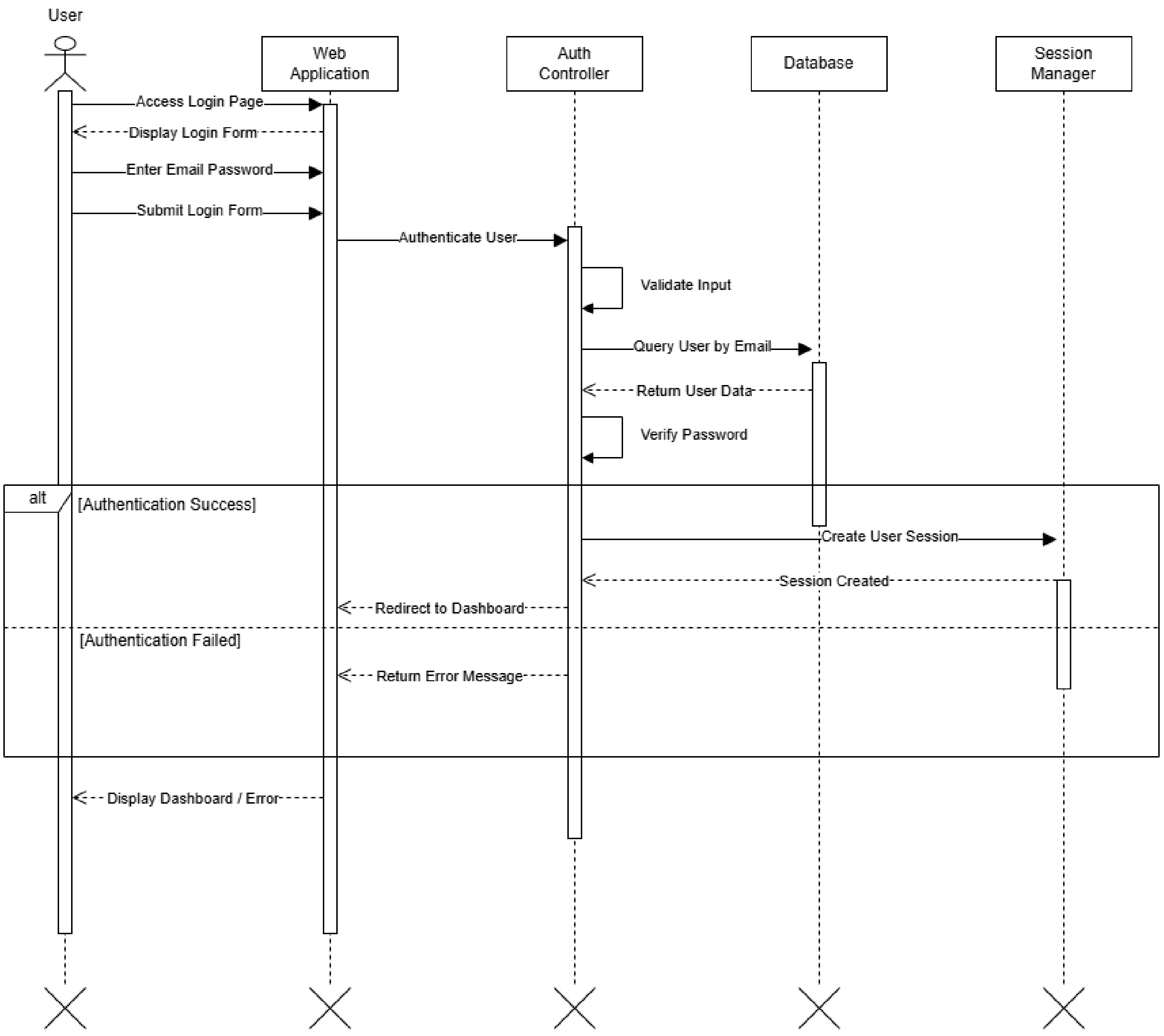

Figure 5: Sequence diagram for user authentication process

3 Implementation

3.1 Technology Stack

Technology Overview

- **Frontend**: HTML5, CSS3, JavaScript ES6, Bootstrap 5.1
- **Backend**: PHP 8.0+, PDO for database operations
- **Database**: MySQL 8.0
- **Server**: Apache (XAMPP for development)
- **Version Control**: Git
- **Development Environment**: Visual Studio Code

3.2 Project Structure

The project follows a logical folder structure that separates concerns and promotes maintainability:

```
advanced-web-dev-exam/
          admin/                  # Administrative interface
                index.php           # Admin dashboard
                events.php          # Event management
                bookings.php        # Booking management
                reports.php         # Analytics and reports
                api/                # Admin API endpoints
          api/                    # Public API endpoints
                get_events.php      # Event retrieval API
                process_order.php   # Order processing
                cart.php            # Cart management
                search_events.php   # Event search functionality
          assets/                 # Static resources
                css/                # Stylesheets
                js/                 # JavaScript files
                images/             # Event images and graphics
          auth/                   # Authentication system
                login.php           # User login
                register.php        # User registration
                logout.php          # Session termination
          config/                 # Configuration files
                database.php        # Database connection setup
          database/               # Database schema and migrations
                schema.sql          # Database structure
          includes/               # Reusable components
                header.php          # Common HTML header
                navbar.php          # Navigation component
                footer.php          # Common HTML footer
          index.php               # Application entry point
          events.php              # Event listing page
          event-details.php       # Individual event details
          cart.php                # Shopping cart interface
          checkout.php            # Checkout process
```

```
        booking-confirmation.php # Booking confirmation
        booking-history.php    # User booking history
```
Listing 1: Project Directory Structure

3.3 Architecture Rationale

3.3.1 Separation of Concerns

The folder structure implements clear separation:

- **/admin**: Contains all administrative functionality, separated from public interfaces
- **/api**: RESTful endpoints that can be consumed by both web interface and potential mobile apps
- **/includes**: Reusable components following DRY (Don't Repeat Yourself) principles
- **/config**: Centralized configuration management

3.3.2 Scalability Considerations

- Modular API design allows for easy integration with mobile applications
- Separate admin interface enables role-based access control
- Asset organization supports CDN integration for future optimization

3.4 Key Implementation Decisions

3.4.1 Database Connection Management

We implemented a singleton pattern for database connections to ensure efficient resource utilization:

```php
<?php
class Database {
    private $host = 'localhost';
    private $db_name = 'event_booking_system';
    private $username = 'dilan';
    private $password = 't92x.7a!lJZEtGjB';
    private $conn;

    public function getConnection() {
        $this->conn = null;
        try {
            $this->conn = new PDO(
                "mysql:host=" . $this->host . ";dbname=" . $this->db_name,
                $this->username,
                $this->password
            );
            $this->conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
        } catch(PDOException $exception) {
            echo "Connection error: " . $exception->getMessage();
        }
        return $this->conn;
    }
}
?>
```

Listing 2: Database Connection Class

Benefits of this approach:

- **Security**: Uses PDO prepared statements to prevent SQL injection
- **Error Handling**: Comprehensive exception handling for database errors
- **Reusability**: Single connection instance shared across the application
- **Configuration**: Centralized database credentials management

3.4.2 Frontend JavaScript Architecture

The frontend uses modular JavaScript with reusable functions:

```
function displayEvents(events, containerId) {
    const container = document.getElementById(containerId);

    if (!events || events.length === 0) {
        container.innerHTML = '<div class="col-12"><div class="alert alert-info">No events found.</div></div>';
        return;
    }

    let html = '';
    events.forEach(event => {
        html += `
            <div class="col-md-4 mb-4">
                <div class="card h-100">
                    <img src="${event.image || 'assets/images/default-event.jpg'}"
                         class="card-img-top" alt="${event.title}"
                         style="height: 200px; object-fit: cover;">
                    <div class="card-body d-flex flex-column">
                        <h5 class="card-title">${event.title}</h5>
                        <p class="card-text">${event.description.substring(0, 100)}...</p>
                        <div class="mt-auto">
                            <div class="d-flex justify-content-between align-items-center mb-2">
                                <small class="text-muted">
                                    <i class="fas fa-calendar me-1"></i>${event.formatted_date}
                                </small>
                                <span class="badge bg-primary">${event.price_formatted}</span>
                            </div>
                            <button class="btn btn-primary w-100" onclick="viewEvent(${event.id})">
                                View Details
                            </button>
                        </div>
                    </div>
                </div>
            </div>
        `;
    });

    container.innerHTML = html;
}
```

Listing 3: Event Display Function

Key features of this implementation:

- **Responsive Design**: Uses Bootstrap grid system for mobile compatibility
- **Error Handling**: Graceful handling of empty or missing data
- **Template Literals**: Modern JavaScript syntax for clean HTML generation
- **Accessibility**: Proper alt text and semantic HTML structure
- **User Experience**: Consistent card layout with hover effects

3.5 Database Schema Implementation

The database schema was designed with normalization principles to ensure data integrity and efficiency:

```sql
CREATE TABLE events (
    id INT PRIMARY KEY AUTO_INCREMENT,
    title VARCHAR(200) NOT NULL,
    description TEXT,
    venue VARCHAR(200) NOT NULL,
    location VARCHAR(200),
    event_date DATE NOT NULL,
    event_time TIME NOT NULL,
    price DECIMAL(10,2) NOT NULL,
    max_capacity INT NOT NULL,
    available_tickets INT,
    total_tickets INT,
    current_bookings INT DEFAULT 0,
    image VARCHAR(500),
    organizer_name VARCHAR(100),
    organizer_contact VARCHAR(100),
    is_featured BOOLEAN DEFAULT FALSE,
    status ENUM('active', 'inactive', 'cancelled', 'full') DEFAULT '
   active',
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE
   CURRENT_TIMESTAMP
);
```

Listing 4: Core Events Table Structure

Design considerations:

- **Data Types**: Appropriate types for each field (DECIMAL for currency, ENUM for status)

- **Constraints**: NOT NULL constraints for essential fields
- **Indexing**: Primary keys and foreign keys for optimal query performance
- **Timestamps**: Automatic tracking of record creation and modification
- **Business Logic**: Status field supports complete event lifecycle management

3.6 Security Implementation

3.6.1 Authentication System

The system implements secure user authentication with the following features:

- **Password Hashing**: Uses PHP's `password_hash()` function with bcrypt
- **Session Management**: Secure session handling with regeneration
- **Input Validation**: Server-side validation for all user inputs
- **SQL Injection Prevention**: PDO prepared statements throughout

3.6.2 Data Protection

- **CSRF Protection**: Token-based protection for forms
- **XSS Prevention**: Output escaping and input sanitization
- **Access Control**: Role-based permissions for admin functions
- **Database Security**: Separate user credentials with minimal privileges

4 Deployment Strategy

4.1 Cloud Deployment Options

For production deployment, several cloud platforms offer suitable environments for this PHP-based application:

4.1.1 Recommended Platform: DigitalOcean

- **LAMP Stack Droplets**: Pre-configured Linux, Apache, MySQL, PHP environment
- **Managed Databases**: Separate database hosting for better performance and backup
- **Load Balancers**: Horizontal scaling capabilities as user base grows
- **Cost Efficiency**: Competitive pricing for African market deployment

4.1.2 Alternative Platforms

- **AWS EC2**: More complex but highly scalable with extensive service ecosystem
- **Google Cloud Platform**: Strong AI/ML integration potential for future features
- **Shared Hosting**: Cost-effective for initial deployment (Namecheap, Hostinger)

4.2 Deployment Architecture

Production Environment

- **Web Server**: Apache 2.4+ with SSL/TLS certificates
- **Database**: MySQL 8.0 with regular backups
- **CDN**: CloudFlare for asset delivery and DDoS protection
- **Monitoring**: Uptime monitoring and error logging
- **Domain**: Custom domain with proper DNS configuration

4.3 Performance Optimization

- **Database Indexing**: Optimized queries with proper indexing
- **Caching**: Session-based caching for frequently accessed data
- **Image Optimization**: Compressed images with appropriate formats
- **Minification**: CSS and JavaScript minification for faster loading

5 Code Documentation

5.1 API Endpoints

The system exposes several RESTful API endpoints for frontend-backend communication:

Endpoint	Method	Purpose	Parameters
/api/get_events.php	GET	Retrieve events	featured, limit, search
/api/process_order.php	POST	Process booking	booking data
/api/cart.php	POST/GET	Cart management	action, event_id
/api/search_events.php	GET	Search events	query, filters

Table 2: Main API Endpoints

5.2 Event Booking Process Flow

The booking process implements a multi-step workflow ensuring data integrity:

```
// 1. Validate user input
$errors = validateBookingData($_POST);
if (!empty($errors)) {
    return json_encode(['success' => false, 'errors' => $errors]);
}

// 2. Begin database transaction
$pdo->beginTransaction();

try {
    // 3. Create booking record
    $booking_id = createBooking($user_data, $total_amount);

    // 4. Process each cart item
    foreach ($cart_items as $item) {
        // Update event capacity
        updateEventCapacity($item['event_id'], $item['quantity']);

        // Create booking item
        createBookingItem($booking_id, $item);
    }

    // 5. Clear user cart
    clearUserCart($user_id);

    // 6. Commit transaction
    $pdo->commit();

    return json_encode(['success' => true, 'booking_id' => $booking_id
   ]);

} catch (Exception $e) {
    // 7. Rollback on error
    $pdo->rollback();
    return json_encode(['success' => false, 'error' => $e->getMessage()
   ]);
}
```

Listing 5: Order Processing Logic (Simplified)

Transaction Management Benefits:

- **Atomicity**: All operations succeed or all fail
- **Consistency**: Database remains in valid state
- **Isolation**: Concurrent bookings don't interfere
- **Durability**: Committed transactions persist

5.3 Frontend-Backend Communication

The application uses AJAX for seamless user experience:

```
function loadFeaturedEvents() {
    fetch('api/get_events.php?featured=true')
        .then(response => {
            if (!response.ok) {
                throw new Error('Network response was not ok');
            }
            return response.json();
        })
        .then(data => {
            displayEvents(data, 'featured-events');
        })
        .catch(error => {
            console.error('Error loading events:', error);
            document.getElementById('featured-events').innerHTML =
                '<div class="alert alert-danger">Error loading events.
    Please try again.</div>';
        });
}
```

Listing 6: AJAX Event Loading

Error Handling Features:

- **Network Error Detection**: Checks response status
- **User Feedback**: Displays meaningful error messages
- **Graceful Degradation**: System remains functional during partial failures
- **Logging**: Client-side error logging for debugging

5.4 Security Measures in Code

5.4.1 Input Sanitization

```
function validateEventInput($data) {
    $errors = [];

    // Title validation
    if (empty(trim($data['title']))) {
        $errors[] = "Event title is required";
    } elseif (strlen($data['title']) > 200) {
        $errors[] = "Event title must be less than 200 characters";
    }

    // Price validation
    if (!is_numeric($data['price']) || $data['price'] < 0) {
        $errors[] = "Valid price is required";
    }

    // Date validation
    if (!validateDate($data['event_date'])) {
        $errors[] = "Valid event date is required";
    }

    return $errors;
}
```

Listing 7: Input Validation Example

5.4.2 Authentication Check

```
function requireLogin() {
    session_start();
    if (!isset($_SESSION['user_id'])) {
        header('Location: auth/login.php');
        exit();
    }
}

function requireAdmin() {
    requireLogin();
    if (!isset($_SESSION['is_admin']) || !$_SESSION['is_admin']) {
        header('Location: index.php');
        exit();
    }
}
```

Listing 8: Session Authentication

5.5 Performance Optimization Code

5.5.1 Database Query Optimization

```
function getEvents($featured = false, $limit = 12, $offset = 0) {
    $sql = "SELECT e.*,
                   DATE_FORMAT(e.event_date, '%M %d, %Y') as
    formatted_date,
                   TIME_FORMAT(e.event_time, '%h:%i %p') as
    formatted_time,
                   CONCAT('XAF ', FORMAT(e.price, 0)) as
    price_formatted
            FROM events e
            WHERE e.status = 'active'";

    if ($featured) {
        $sql .= " AND e.is_featured = 1";
    }

    $sql .= " ORDER BY e.event_date ASC, e.event_time ASC
              LIMIT :limit OFFSET :offset";

    $stmt = $this->pdo->prepare($sql);
    $stmt->bindValue(':limit', $limit, PDO::PARAM_INT);
    $stmt->bindValue(':offset', $offset, PDO::PARAM_INT);
    $stmt->execute();

    return $stmt->fetchAll(PDO::FETCH_ASSOC);
}
```

Listing 9: Optimized Event Retrieval

Optimization techniques used:

- **Formatted Fields**: Database-level formatting reduces PHP processing
- **Prepared Statements**: Prevent SQL injection and improve performance
- **Pagination**: LIMIT and OFFSET for efficient large dataset handling
- **Indexing**: Database indexes on commonly queried fields

6 Future Enhancements

6.1 Planned Features

- **Mobile Application**: React Native or Flutter mobile app
- **Payment Integration**: MTN Mobile Money and Orange Money integration
- **Email Notifications**: Automated booking confirmations and reminders
- **Social Features**: Event sharing and social media integration
- **Analytics Dashboard**: Advanced reporting and business intelligence
- **Multi-language Support**: French and local language translations

6.2 Scalability Considerations

- **Microservices Architecture**: Break down into smaller, manageable services
- **Caching Layer**: Redis implementation for improved performance
- **Queue System**: Background job processing for heavy operations
- **API Versioning**: Support multiple API versions for backward compatibility

7 Conclusion

The EventBook system represents a comprehensive solution for event management and booking in the Cameroonian market. The implementation demonstrates solid software engineering principles, security best practices, and scalable architecture design.

Key achievements include:

- **Full-stack Implementation**: Complete web application with frontend and backend
- **Security Focus**: Comprehensive security measures throughout the system
- **Cultural Relevance**: Tailored specifically for Cameroonian events and culture
- **Scalable Design**: Architecture supports future growth and feature additions
- **Modern Technologies**: Uses current web development standards and best practices

The system is ready for production deployment and provides a solid foundation for future enhancements and market expansion.