ShuttleSync

Railway Management System

Project Documentation

August 15, 2025

Name	ID
Afia Tasnim Tahura	20701034
Jannatul Maoua Saima	20701081
Muhd. Rifat Uddin	21701017

Table 1: Project Members

Abstract

ShuttleSync is a comprehensive railway management system built using the MERN (MongoDB, Express.js, React.js, Node.js) stack. The application provides a complete solution for railway ticket booking, user authentication, train search functionality, secure payment processing, booking management, and real-time notifications. This documentation covers the system architecture, features, implementation details, and usage instructions for the ShuttleSync platform.

Contents

1 Introduction

1.1 Project Overview

ShuttleSync is a modern web-based railway management system designed to streamline the train booking process for passengers while providing administrators with efficient management tools. The system leverages cutting-edge web technologies to deliver a seamless user experience with robust security measures and real-time functionality.

1.2 Objectives

- Provide a user-friendly interface for train booking and management
- Implement secure authentication and authorization mechanisms
- Enable efficient train search and booking functionality
- Integrate secure payment processing with multiple scenarios
- Deliver real-time notifications and updates to users
- Maintain comprehensive booking records and history

1.3 Scope

The ShuttleSync system encompasses:

- User registration and authentication system
- Train search and availability checking
- Secure booking and payment processing
- User dashboard for booking management
- Administrative features for system management
- Real-time notification system

2 System Architecture

2.1 Technology Stack

The ShuttleSync system is built using the MERN stack architecture:

Component	Technology
Frontend	React.js with modern hooks and context API
Backend	Node.js with Express.js framework
Database	MongoDB with Mongoose ODM
Authentication	JSON Web Tokens (JWT)
Payment	Integrated payment gateway
Real-time	WebSocket/Socket.io for live notifications

Table 2: Technology Stack Components

2.2 Architecture Pattern

The system follows a three-tier architecture:

1. **Presentation Layer**: React.js frontend providing user interface
2. **Business Logic Layer**: Express.js backend handling application logic
3. **Data Access Layer**: MongoDB database for data persistence

2.3 System Components

- **Frontend (React.js)**: Single Page Application with component-based architecture
- **Backend API (Express.js)**: RESTful API services for data operations
- **Database (MongoDB)**: NoSQL database for flexible data storage
- **Authentication Service**: JWT-based secure authentication system
- **Payment Service**: Integrated payment gateway with error handling
- **Notification Service**: Real-time update delivery system

3 Features and Functionality

3.1 User Authentication System

3.1.1 Registration Process

- User account creation with email validation
- Password encryption using bcrypt hashing
- Form validation and error handling
- Automatic JWT token generation upon successful registration

3.1.2 Login System

- Secure credential verification
- JWT token-based session management
- Remember me functionality
- Password reset capabilities

3.2 Train Search and Booking

3.2.1 Search Functionality

- Origin and destination station selection
- Date-based availability checking
- Real-time seat availability display
- Filter options for train types and classes

3.2.2 Booking Process

- Seat selection interface
- Passenger information collection
- Booking confirmation with unique reference number
- PDF ticket generation

3.3 Payment Gateway Integration

3.3.1 Payment Processing

- Multiple payment method support
- Secure transaction processing
- Payment verification and confirmation

3.3.2 Payment Status Handling

- **Successful Payment**: Automatic booking confirmation and ticket generation
- **Payment Failure**: Error handling with retry options and booking cancellation
- Transaction logging for audit purposes

3.4 Booking Management

- Personal booking history display
- Booking details and status tracking
- Cancellation and modification options
- Digital ticket download functionality

3.5 Real-time Notifications

- Live train schedule updates
- Platform and gate change announcements
- Delay notifications
- Emergency notices and alerts
- Booking confirmation messages

4 Database Design

4.1 Data Models

4.1.1 User Schema

```
const userSchema = {
  _id: ObjectId,
  firstName: String,
  lastName: String,
  email: String (unique),
  password: String (hashed),
  phone: String,
  dateOfBirth: Date,
  createdAt: Date,
  updatedAt: Date
}
```

Listing 1: User Model Structure

4.1.2 Train Schema

```
const trainSchema = {
  _id: ObjectId,
  trainNumber: String (unique),
  trainName: String,
  origin: String,
  destination: String,
  departureTime: String,
  arrivalTime: String,
  totalSeats: Number,
  availableSeats: Number,
```

```
  fare: Number,
  trainType: String
}
```

Listing 2: Train Model Structure

4.1.3 Booking Schema

```
const bookingSchema = {
  _id: ObjectId,
  userId: ObjectId,
  trainId: ObjectId,
  bookingReference: String (unique),
  passengerDetails: Array,
  seatNumbers: Array,
  journeyDate: Date,
  bookingStatus: String,
  paymentStatus: String,
  totalAmount: Number,
  createdAt: Date
}
```

Listing 3: Booking Model Structure

5 API Documentation

5.1 Authentication Endpoints

Method	Endpoint	Description	Auth Required
POST	/api/auth/register	User registration	No
POST	/api/auth/login	User login	No
POST	/api/auth/logout	User logout	Yes
GET	/api/auth/profile	Get user profile	Yes

Table 3: Authentication API Endpoints

5.2 Train and Booking Endpoints

Method	Endpoint	Description	Auth Required
GET	/api/trains/search	Search trains	No
GET	/api/trains/:id	Get train details	No
POST	/api/bookings/create	Create new booking	Yes
GET	/api/bookings/user	Get user bookings	Yes
PUT	/api/bookings/:id/cancel	Cancel booking	Yes

Table 4: Train and Booking API Endpoints

6 Security Implementation

6.1 Authentication Security

- JWT tokens with expiration times
- Password hashing using bcrypt with salt rounds
- Protected routes with middleware authentication
- Session management and token refresh mechanisms

6.2 Data Validation

- Input sanitization to prevent injection attacks
- Schema validation using Joi or similar libraries
- CORS configuration for cross-origin requests
- Rate limiting to prevent abuse

6.3 Payment Security

- Secure payment gateway integration
- Transaction encryption
- PCI compliance considerations
- Fraud detection mechanisms

7 Installation and Setup

7.1 Prerequisites

- Node.js (version 14.x or higher)
- MongoDB (version 4.x or higher)
- npm or yarn package manager
- Git for version control

7.2 Installation Steps

1. Clone the repository:

```
git clone https://github.com/yourusername/shuttlesync.git
cd shuttlesync
```

2. Install backend dependencies:

```
cd backend
npm install
```

3. Install frontend dependencies:

```
cd ../frontend
npm install
```

4. Configure environment variables:

```
# Create .env file in backend directory
cp .env.example .env
# Edit .env with your configuration
```

5. Start the application:

```
# Start backend server
cd backend
npm start

# In another terminal, start frontend
cd frontend
npm start
```

7.3 Environment Configuration

```
# Database Configuration
MONGODB_URI=mongodb://localhost:27017/shuttlesync
DB_NAME=shuttlesync

# JWT Configuration
JWT_SECRET=your-secret-key-here
JWT_EXPIRE=24h

# Payment Gateway
PAYMENT_GATEWAY_KEY=your-payment-key
PAYMENT_GATEWAY_SECRET=your-payment-secret

# Application Configuration
PORT=5000
NODE_ENV=development
CLIENT_URL=http://localhost:3000
```

Listing 4: Environment Variables (.env)

8 Usage Guide

8.1 User Registration and Login

1. Navigate to the registration page
2. Fill in required personal information

3. Verify email address if required
4. Log in using registered credentials
5. Access the main dashboard upon successful authentication

8.2 Train Search and Booking

1. Select origin and destination stations
2. Choose travel date
3. Browse available trains and timings
4. Select preferred train and seat class
5. Proceed to booking with passenger details
6. Complete payment process
7. Receive booking confirmation and digital ticket

8.3 Managing Bookings

1. Access "My Bookings" section from user dashboard
2. View booking history and current reservations
3. Download digital tickets
4. Cancel or modify bookings as needed
5. Track booking status and updates

9 Testing and Quality Assurance

9.1 Testing Strategy

- **Unit Testing**: Individual component and function testing
- **Integration Testing**: API endpoint and database interaction testing
- **End-to-End Testing**: Complete user workflow testing
- **Security Testing**: Authentication and authorization validation

9.2 Performance Considerations

- Database query optimization
- Frontend component lazy loading
- API response caching
- Image and asset optimization

10 Future Enhancements

10.1 Planned Features

- Mobile application development
- Advanced analytics and reporting
- Multi-language support
- Integration with external railway APIs
- Enhanced notification system with SMS and email

10.2 Scalability Improvements

- Microservices architecture migration
- Load balancing implementation
- Database sharding for large-scale data
- CDN integration for better performance

11 Conclusion

ShuttleSync represents a comprehensive solution for modern railway management needs. The system successfully combines user-friendly design with robust backend functionality, providing a seamless experience for both passengers and administrators. The implementation of modern web technologies ensures scalability, security, and maintainability. The project demonstrates proficiency in full-stack development, database design, security implementation, and system integration. The modular architecture allows for easy maintenance and future enhancements, making ShuttleSync a viable solution for real-world railway management scenarios.

12 References

1. MongoDB Documentation. *MongoDB Manual*. MongoDB Inc., 2024.
2. Express.js Guide. *Express.js Documentation*. Node.js Foundation, 2024.
3. React.js Documentation. *React - A JavaScript Library for Building User Interfaces*. Meta, 2024.
4. JSON Web Tokens. *JWT Introduction*. Auth0, 2024.
5. Node.js Documentation. *Node.js API Reference*. Node.js Foundation, 2024.