Software Design Specifications

for

Hostel Management System v1.0

Prepared by:
Nandan Murari
SE22UARI110

Document Information

Title: Hostel Management System Software Design Specification

Project Manager: Nandan Murari

Document Version No: 1.0

Document Version Date: April 9, 2025

Prepared By: Nandan Murari

Preparation Date: April 9, 2025

Version History

Ver. No.	Ver. Date	Revised By	Description	Filename
1.0	09-04-2025	Nandan Murari	Initial Draft	HMS_SDS_v1.0.doc

Table of Contents

1. INTRODUCTION 1.1 PURPOSE 1.2 SCOPE 1.3 DEFINITIONS, ACRONYMS, AND ABBREVIATIONS 1.4 REFERENCES
2. USE CASE VIEW 2.1 USE CASES
3. DESIGN OVERVIEW 3.1 DESIGN GOALS AND CONSTRAINTS 3.2 DESIGN ASSUMPTIONS 3.3 SIGNIFICANT DESIGN PACKAGES 3.4 DEPENDENT EXTERNAL INTERFACES 3.5 IMPLEMENTED APPLICATION EXTERNAL INTERFACES
4. LOGICAL VIEW 4.1 DESIGN MODEL 4.2 USE CASE REALIZATION
5. DATA VIEW 5.1 DOMAIN MODEL 5.2 DATA MODEL (PERSISTENT DATA VIEW) 5.2.1 DATA DICTIONARY

6. EXCEPTION HANDLING
7. CONFIGURABLE PARAMETERS
8. QUALITY OF SERVICE 8.1 AVAILABILITY 8.2 SECURITY AND AUTHORIZATION 8.3 LOAD AND PERFORMANCE IMPLICATIONS 8.4 MONITORING AND CONTROL

1 Introduction

This Software Design Specification (SDS) provides a comprehensive technical description of the Hostel Management System. It serves as a blueprint for implementing a system that enables seamless interaction between parents, students, and wardens, managing crucial aspects like fee payments, room allotments, and bookings.

1.1 Purpose

The purpose of this SDS is to outline the detailed design of the Hostel Management System to guide developers through the implementation process. This document provides an overview of the system architecture, component interactions, and technical specifications. It serves as a reference for the development team, project stakeholders, and future maintenance personnel.

1.2 Scope

This SDS applies to the Hostel Management System v1.0, encompassing all modules required for managing hostel operations, including:

- User authentication and authorization for students, parents, and wardens
- Room inventory management and allocation
- Fee calculation, payment processing, and tracking
- Communication channels between stakeholders
- Complaint and feedback management
- Reporting and analytics

The document describes the technical architecture, database design, user interfaces, and system integration points needed to implement these features.

1.3 Definitions, Acronyms, and Abbreviations

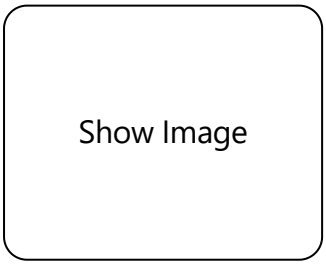
Term/Acronym	Definition
HMS	Hostel Management System
UI	User Interface
API	Application Programming Interface
SRS	Software Requirements Specification
SDS	Software Design Specification
DBMS	Database Management System
HTTP	Hypertext Transfer Protocol
CRUD	Create, Read, Update, Delete
REST	Representational State Transfer
JSON	JavaScript Object Notation
JWT	JSON Web Token
MVC	Model-View-Controller
RBAC	Role-Based Access Control

1.4 References

1. Software Requirements Specification (SRS) for Hostel Management System v1.0
2. University Hostel Management Policy Document
3. MySQL 8.0 Documentation: https://dev.mysql.com/doc/
4. JavaScript ECMAScript 2021 Specification
5. HTML5 and CSS3 Specifications: https://www.w3.org/TR/html5/
6. Payment Gateway Integration Documentation

2 Use Case View

2.1 Use Cases



2.1.1 Authentication and User Management

UC-1: User Registration

- Brief Description: Allows new users (students and parents) to register in the system
- Primary Actors: Students, Parents

- Preconditions: User is not registered
- Main Flow:
    1. User navigates to registration page
    2. System displays registration form
    3. User provides required information and submits
    4. System validates information
    5. System creates user account
    6. System sends verification email
    7. User verifies email
    8. System activates account
- Post-conditions: User account is created and activated

UC-2: User Login

- Brief Description: Authenticates users to access the system
- Primary Actors: Students, Parents, Wardens, Administrators
- Preconditions: User is registered
- Main Flow:
    1. User navigates to login page
    2. User enters credentials
    3. System validates credentials
    4. System generates session token
    5. System redirects to appropriate dashboard
- Post-conditions: User is logged in and granted appropriate access

UC-3: Password Reset

- Brief Description: Allows users to reset forgotten passwords
- Primary Actors: All users
- Preconditions: User is registered
- Main Flow:
    1. User requests password reset
    2. System sends reset link to registered email
    3. User follows link and sets new password
    4. System updates password
- Post-conditions: User's password is updated

2.1.2 Room Management

UC-4: View Available Rooms

- Brief Description: Displays available rooms based on search criteria
- Primary Actors: Students
- Preconditions: User is logged in as student
- Main Flow:
  1. Student navigates to room search page
  2. Student sets search filters (floor, room type, etc.)
  3. System displays matching available rooms
- Post-conditions: Available rooms are displayed to student

UC-5: Room Booking

- Brief Description: Allows students to book available rooms
- Primary Actors: Students
- Preconditions: User is logged in as student, rooms are available
- Main Flow:
  1. Student selects a room to book
  2. System displays booking details and terms
  3. Student confirms booking
  4. System reserves room temporarily
  5. Student is directed to payment page
- Post-conditions: Room is temporarily reserved for the student

UC-6: Room Allocation

- Brief Description: Warden allocates rooms to students
- Primary Actors: Warden
- Preconditions: Warden is logged in, rooms are available
- Main Flow:
  1. Warden searches for student
  2. Warden views available rooms
  3. Warden selects room to allocate
  4. Warden confirms allocation
  5. System updates room status

6. System notifies student

- Post-conditions: Room is allocated to student

2.1.3 Fee Management

UC-7: Fee Payment

- Brief Description: Enables payment of hostel fees
- Primary Actors: Students, Parents
- Preconditions: User is logged in, invoice is generated
- Main Flow:
  1. User navigates to payments page
  2. User selects invoice to pay
  3. User chooses payment method
  4. System redirects to payment gateway
  5. User completes payment
  6. System receives payment confirmation
  7. System updates payment status
  8. System generates receipt
- Post-conditions: Payment is recorded, receipt is generated

UC-8: View Payment History

- Brief Description: Displays history of fee payments
- Primary Actors: Students, Parents, Wardens
- Preconditions: User is logged in
- Main Flow:
  1. User navigates to payment history page
  2. System retrieves payment records
  3. System displays payment history
- Post-conditions: Payment history is displayed

2.1.4 Communication

UC-9: Send Notification

- Brief Description: Allows warden to send notifications to students/parents
- Primary Actors: Warden
- Preconditions: Warden is logged in

- Main Flow:
    1. Warden navigates to notification page
    2. Warden selects recipients (individual/group)
    3. Warden composes message
    4. Warden sends notification
    5. System delivers notification to recipients
- Post-conditions: Notification is sent to recipients

UC-10: Submit Complaint/Request

- Brief Description: Allows students to submit complaints or requests
- Primary Actors: Students
- Preconditions: Student is logged in
- Main Flow:
    1. Student navigates to complaint/request page
    2. Student fills complaint/request form
    3. Student submits form
    4. System records complaint/request
    5. System notifies warden
- Post-conditions: Complaint/request is recorded, warden is notified

3 Design Overview

3.1 Design Goals and Constraints

3.1.1 Design Goals

1. **User-Friendly Interface**: Design intuitive interfaces for all user types, with special consideration for parents who may be less tech-savvy.
2. **Scalability**: Support a growing number of users and rooms without performance degradation.
3. **Security**: Implement robust security measures to protect sensitive data, especially financial transactions.
4. **Modularity**: Create loosely coupled components to facilitate maintenance and future enhancements.
5. **Responsiveness**: Ensure the system is accessible from various devices and screen sizes.
6. **Real-time Updates**: Provide real-time status updates for room availability and fee payments.
7. **Maintainability**: Ensure the code is well-structured and documented for easy maintenance.

3.1.2 Constraints

1. **Technology Stack**: The system must be developed using JavaScript, MySQL, HTML, and CSS.
2. **Browser Compatibility**: The system must support modern web browsers (Chrome, Firefox, Edge, Safari).
3. **Data Protection**: The system must comply with relevant data protection regulations.
4. **Integration Limitations**: Integration capabilities may be constrained by the APIs available from third-party services.
5. **Performance**: The system must handle concurrent user sessions without significant performance degradation.
6. **Development Timeline**: The system must be completed within the specified project timeline.

3.2 Design Assumptions

1. All users have basic computer literacy and internet access.
2. The system will be deployed on a web server with adequate processing power, memory, and storage.
3. The database server will have sufficient capacity to handle the expected data volume and transaction load.
4. Payment gateway services will be available and reliable.
5. Users will access the system primarily through web browsers on computers and mobile devices.
6. There will be administrative support for initial data setup and maintenance.
7. The institution has a stable internet connection capable of supporting the system's bandwidth requirements.
8. Each room has a unique identifier and fixed characteristics (size, capacity, etc.).

3.3 Significant Design Packages

The Hostel Management System follows a modular design with the following major packages:

1. **User Management Module**
   - User Authentication
   - User Registration
   - Profile Management
   - Role-Based Access Control
2. **Room Management Module**
   - Room Inventory
   - Room Search and Filtering
   - Booking Management

- Allocation Management

3. **Fee Management Module**
   - Fee Calculation
   - Invoice Generation
   - Payment Processing
   - Receipt Generation
   - Payment History
4. **Communication Module**
   - Notification System
   - Complaint/Request Management
   - Feedback System
   - Messaging System
5. **Reporting Module**
   - Occupancy Reports
   - Financial Reports
   - Student Reports
   - Audit Logs
6. **Administration Module**
   - System Configuration
   - User Administration
   - Data Management
   - Backup and Recovery

3.4 Dependent External Interfaces

External Interface	Module Using the Interface	Description
Payment Gateway API	Fee Management Module	Interface for processing online payments through third-party payment providers
Email Service API	Communication Module, User Management Module	Interface for sending emails for notifications, account verification, password resets
SMS Gateway API	Communication Module	Interface for sending SMS notifications to parents and students
University Student Database API	User Management Module	Interface for validating student information during registration
Document Storage Service	Room Management, Fee Management	Interface for storing and retrieving documents such as receipts and agreements

3.5 Implemented Application External Interfaces (and SOA web services)

Interface Name	Module Implementing the Interface	Description
User Authentication API	User Management Module	RESTful API for user authentication, registration, and profile management
Room Management API	Room Management Module	RESTful API for room search, booking, and allocation operations
Payment API	Fee Management Module	RESTful API for fee calculation, payment processing, and history retrieval
Notification API	Communication Module	RESTful API for sending and managing notifications to users
Reporting API	Reporting Module	RESTful API for generating and retrieving various system reports

4 Logical View

4.1 Design Model

The Hostel Management System follows the Model-View-Controller (MVC) architectural pattern, which separates the application into three main components:

4.1.1 Model Layer

The model layer represents the application's data structure and business logic.

Key Classes:

1. **User Class**
   - Properties: userId, userName, email, passwordHash, role, contactNumber, profilePicture, status
   - Methods: authenticate(), updateProfile(), resetPassword(), getNotifications()
2. **Student Class (extends User)**
   - Properties: studentId, rollNumber, department, year, parentId, roomId
   - Methods: bookRoom(), payFees(), submitRequest(), viewPaymentHistory()
3. **Parent Class (extends User)**
   - Properties: parentId, studentIds[]
   - Methods: viewChildrenProfile(), payFees(), viewPaymentHistory()
4. **Warden Class (extends User)**
   - Properties: employeeId, designation
   - Methods: allocateRoom(), approveRequest(), sendNotification(), generateReport()
5. **Room Class**
   - Properties: roomId, roomNumber, floor, building, capacity, type, description, status, amenities[], currentOccupants[]
   - Methods: checkAvailability(), allocate(), deallocate(), updateStatus()
6. **Booking Class**
   - Properties: bookingId, studentId, roomId, bookingDate, status, paymentStatus
   - Methods: confirm(), cancel(), updateStatus(), getDetails()
7. **Payment Class**
   - Properties: paymentId, studentId, amount, paymentDate, paymentMethod, transactionId, status
   - Methods: process(), generateReceipt(), verifyStatus(), refund()
8. **Invoice Class**

- Properties: invoiceId, studentId, amount, dueDate, status, items[]
- Methods: generate(), update(), markAsPaid()

9. **Notification Class**
    - Properties: notificationId, senderId, recipientIds[], title, message, sentDate, status
    - Methods: send(), markAsRead(), delete()
10. **Request/Complaint Class**
    - Properties: requestId, studentId, type, title, description, submissionDate, status, resolutionDetails
    - Methods: submit(), update(), resolve(), escalate()

4.1.2 View Layer

The view layer consists of HTML templates and CSS for rendering the user interface.

Key Views:

1. **Authentication Views**
    - Login Page
    - Registration Page
    - Forgot Password Page
    - Reset Password Page
2. **Dashboard Views**
    - Student Dashboard
    - Parent Dashboard
    - Warden Dashboard
    - Admin Dashboard
3. **Room Management Views**
    - Room Search Page
    - Room Details Page
    - Room Booking Page
    - Room Allocation Page
4. **Fee Management Views**
    - Invoice List Page
    - Payment Page
    - Payment History Page
    - Receipt Page

5. **Communication Views**
    - Notification List Page
    - Notification Creation Page
    - Request/Complaint Submission Page
    - Request/Complaint List Page
6. **Report Views**
    - Occupancy Report Page
    - Financial Report Page
    - Student Report Page

4.1.3 Controller Layer

The controller layer handles user requests and orchestrates the interactions between the Model and View layers.

Key Controllers:

1. **AuthController**
    - Methods: login(), register(), forgotPassword(), resetPassword(), logout()
2. **UserController**
    - Methods: getProfile(), updateProfile(), changePassword(), getUsers(), updateUser()
3. **RoomController**
    - Methods: searchRooms(), getRoom(), bookRoom(), allocateRoom(), deallocateRoom()
4. **BookingController**
    - Methods: createBooking(), getBooking(), updateBooking(), cancelBooking(), getBookingHistory()
5. **PaymentController**
    - Methods: initiatePayment(), processPayment(), getPayment(), getPaymentHistory(), generateReceipt()
6. **InvoiceController**
    - Methods: createInvoice(), getInvoice(), updateInvoice(), getInvoices()
7. **NotificationController**
    - Methods: sendNotification(), getNotifications(), markAsRead(), deleteNotification()
8. **RequestController**
    - Methods: submitRequest(), getRequest(), updateRequest(), resolveRequest(), getRequests()
9. **ReportController**
    - Methods: generateOccupancyReport(), generateFinancialReport(), generateStudentReport()

4.2 Use Case Realization

4.2.1 UC-5: Room Booking

Sequence Diagram:

1. Student (Actor) -> RoomController: searchRooms(criteria)
2. RoomController -> Room: findAvailable(criteria)
3. Room -> RoomController: availableRooms[]
4. RoomController -> Student: display available rooms
5. Student -> RoomController: selectRoom(roomId)
6. RoomController -> Room: getDetails(roomId)
7. Room -> RoomController: roomDetails
8. RoomController -> Student: display room details and booking form
9. Student -> RoomController: confirmBooking(roomId, details)
10. RoomController -> Booking: create(studentId, roomId, details)
11. Booking -> RoomController: bookingId
12. RoomController -> Room: updateStatus(roomId, 'reserved')
13. RoomController -> PaymentController: createInvoice(bookingId, studentId)
14. PaymentController -> Invoice: generate(studentId, amount, items)
15. Invoice -> PaymentController: invoiceId
16. PaymentController -> RoomController: invoiceId
17. RoomController -> Student: redirect to payment page

4.2.2 UC-7: Fee Payment

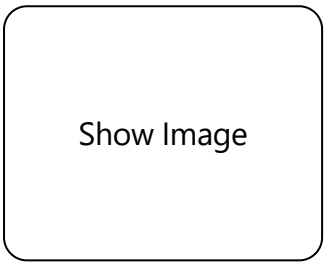
Sequence Diagram:

1. User (Actor) -> PaymentController: viewInvoices()
2. PaymentController -> Invoice: getByUser(userId)
3. Invoice -> PaymentController: invoices[]
4. PaymentController -> User: display invoices
5. User -> PaymentController: selectInvoice(invoiceId)
6. PaymentController -> Invoice: getDetails(invoiceId)
7. Invoice -> PaymentController: invoiceDetails
8. PaymentController -> User: display payment options
9. User -> PaymentController: initiatePayment(invoiceId, method)

10. PaymentController -> Payment Gateway API: processPayment(details)
11. Payment Gateway API -> PaymentController: transactionResult
12. PaymentController -> Payment: create(userId, invoiceId, amount, transactionId)
13. PaymentController -> Invoice: updateStatus(invoiceId, 'paid')
14. PaymentController -> User: display payment confirmation
15. PaymentController -> NotificationController: sendNotification(studentId, parentId, 'Payment Confirmation')

5 Data View

5.1 Domain Model



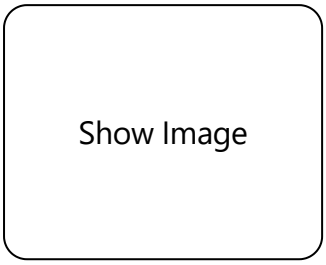
The domain model represents the key entities in the Hostel Management System and their relationships:

1. **User**: Base entity for all system users
   - Specialized into: Student, Parent, Warden, Administrator
2. **Room**: Represents a physical room in the hostel
   - Associated with: Building, Floor
   - Can have multiple: Amenities
3. **Booking**: Represents a room booking request/transaction
   - Associated with: Student, Room
4. **Allocation**: Represents an actual room allocation
   - Associated with: Student, Room, Warden
5. **Invoice**: Represents a fee invoice
   - Associated with: Student
   - Can have multiple: InvoiceItems
6. **Payment**: Represents a payment transaction
   - Associated with: Invoice, Student/Parent
7. **Notification**: Represents system or user-generated notifications
   - Associated with: Sender (User), Recipients (Users)
8. **Request/Complaint**: Represents a request or complaint submitted by a student

- Associated with: Student, Warden (resolver)

5.2 Data Model (Persistent Data View)

The data model describes the database schema for the Hostel Management System using MySQL as the DBMS.



Database Tables:

1. **users**
   - user_id (PK)
   - username
   - email
   - password_hash
   - role
   - contact_number
   - profile_picture
   - status
   - created_at
   - updated_at
2. **students**
   - student_id (PK)
   - user_id (FK)
   - roll_number
   - department
   - year
   - parent_id (FK)
   - room_id (FK)
   - created_at
   - updated_at
3. **parents**
   - parent_id (PK)

- user_id (FK)
- created_at
- updated_at

4. **wardens**
   - warden_id (PK)
   - user_id (FK)
   - employee_id
   - designation
   - created_at
   - updated_at
5. **buildings**
   - building_id (PK)
   - name
   - description
   - floors
   - address
   - created_at
   - updated_at
6. **rooms**
   - room_id (PK)
   - room_number
   - building_id (FK)
   - floor
   - capacity
   - type
   - description
   - status
   - monthly_fee
   - created_at
   - updated_at
7. **room_amenities**
   - amenity_id (PK)
   - room_id (FK)

- amenity_name
- description
- created_at
- updated_at

8. **bookings**
   - booking_id (PK)
   - student_id (FK)
   - room_id (FK)
   - booking_date
   - status
   - payment_status
   - created_at
   - updated_at
9. **allocations**
   - allocation_id (PK)
   - student_id (FK)
   - room_id (FK)
   - warden_id (FK)
   - allocation_date
   - start_date
   - end_date
   - status
   - created_at
   - updated_at
10. **invoices**
    - invoice_id (PK)
    - student_id (FK)
    - amount
    - due_date
    - status
    - created_at
    - updated_at
11. **invoice_items**

- item_id (PK)
- invoice_id (FK)
- description
- amount
- created_at
- updated_at

12. **payments**
    - payment_id (PK)
    - invoice_id (FK)
    - user_id (FK)
    - amount
    - payment_date
    - payment_method
    - transaction_id
    - status
    - created_at
    - updated_at
13. **notifications**
    - notification_id (PK)
    - sender_id (FK)
    - title
    - message
    - sent_date
    - created_at
    - updated_at
14. **notification_recipients**
    - id (PK)
    - notification_id (FK)
    - recipient_id (FK)
    - read_status
    - read_date
    - created_at
    - updated_at

15. **requests**
    - request_id (PK)
    - student_id (FK)
    - type
    - title
    - description
    - submission_date
    - status
    - resolver_id (FK)
    - resolution_details
    - resolution_date
    - created_at
    - updated_at

5.2.1 Data Dictionary

users Table

Field	Type	Description	Constraints
user_id	INT	Unique identifier for each user	Primary Key, Auto Increment
username	VARCHAR(50)	User's username	Not Null, Unique
email	VARCHAR(100)	User's email address	Not Null, Unique
password_hash	VARCHAR(255)	Hashed password	Not Null
role	ENUM	User role (student, parent, warden, admin)	Not Null
contact_number	VARCHAR(20)	User's contact number	Not Null
profile_picture	VARCHAR(255)	Path to profile picture	Nullable
status	ENUM	Account status (active, inactive, suspended)	Not Null, Default 'active'
created_at	TIMESTAMP	Record creation timestamp	Not Null, Default CURRENT_TIMESTAMP
updated_at	TIMESTAMP	Record update timestamp	Not Null, Default CURRENT_TIMESTAMP ON UPDATE

students Table

Field	Type	Description	Constraints
student_id	INT	Unique identifier for each student	Primary Key, Auto Increment
user_id	INT	Reference to users table	Foreign Key, Not Null
roll_number	VARCHAR(20)	Student's roll number	Not Null, Unique
department	VARCHAR(50)	Student's department	Not Null
year	INT	Student's year of study	Not Null
parent_id	INT	Reference to parents table	Foreign Key, Nullable
room_id	INT	Reference to rooms table	Foreign Key, Nullable
created_at	TIMESTAMP	Record creation timestamp	Not Null, Default CURRENT_TIMESTAMP
updated_at	TIMESTAMP	Record update timestamp	Not Null, Default CURRENT_TIMESTAMP ON UPDATE

rooms Table

Field	Type	Description	Constraints
room_id	INT	Unique identifier for each room	Primary Key, Auto Increment
room_number	VARCHAR(20)	Room number	Not Null
building_id	INT	Reference to buildings table	Foreign Key, Not Null
floor	INT	Floor number	Not Null
capacity	INT	Room capacity	Not Null
type	ENUM	Room type (single, double, shared)	Not Null
description	TEXT	Room description	Nullable
status	ENUM	Room status (available, reserved, occupied, maintenance)	Not Null, Default 'available'
monthly_fee	DECIMAL(10,2)	Monthly fee for the room	Not Null
created_at	TIMESTAMP	Record creation timestamp	Not Null, Default CURRENT_TIMESTAMP
updated_at	TIMESTAMP	Record update timestamp	Not Null, Default CURRENT_TIMESTAMP ON UPDATE

payments Table

Field	Type	Description	Constraints
payment_id	INT	Unique identifier for each payment	Primary Key, Auto Increment
invoice_id	INT	Reference to invoices table	Foreign Key, Not Null
user_id	INT	Reference to users table	Foreign Key, Not Null
amount	DECIMAL(10,2)	Payment amount	Not Null
payment_date	DATE	Date of payment	Not Null
payment_method	ENUM	Method of payment (credit_card, debit_card, net_banking, wallet)	Not Null
transaction_id	VARCHAR(100)	External transaction reference	Not Null
status	ENUM	Payment status (pending, successful, failed, refunded)	Not Null, Default 'pending'
created_at	TIMESTAMP	Record creation timestamp	Not Null, Default CURRENT_TIMESTAMP
updated_at	TIMESTAMP	Record update timestamp	Not Null, Default CURRENT_TIMESTAMP ON UPDATE

6 Exception Handling

The Hostel Management System implements a robust exception handling mechanism to manage errors gracefully and provide appropriate feedback to users.

6.1 Global Exception Handler

A global exception handler intercepts all unhandled exceptions in the application and processes them appropriately:

javascript

```javascript
// Global exception handler
app.use((error, req, res, next) => {
  const status = error.statusCode || 500;
  const message = error.message || 'An unexpected error occurred';
  const data = error.data;

  // Log error for debugging
  console.error(`[ERROR] ${status}: ${message}`, error);

  // Return appropriate response to client
  res.status(status).json({
    success: false,
    message: message,
    data: data
  });
});
```

6.2 Custom Error Classes

The system defines several custom error classes to handle specific error scenarios:

javascript

```
// Authentication error
class AuthError extends Error {
  constructor(message) {
    super(message);
    this.statusCode = 401;
    this.name = 'AuthError';
  }
}

// Authorization error
class ForbiddenError extends Error {
  constructor(message) {
    super(message);
    this.statusCode = 403;
    this.name = 'ForbiddenError';
  }
}

// Resource not found error
class NotFoundError extends Error {
  constructor(message) {
    super(message);
    this.statusCode = 404;
    this.name = 'NotFoundError';
  }
}

// Validation error
class ValidationError extends Error {
  constructor(message, data) {
    super(message);
    this.statusCode = 422;
    this.name = 'ValidationError';
    this.data = data;
  }
}

// Payment processing error
class PaymentError extends Error {
  constructor(message) {
    super(message);
    this.statusCode = 500;
    this.name = 'PaymentError';
  }
}
```

6.3 Error Logging

Errors are logged to help with debugging and monitoring:

1. All errors are logged to the console for development environments
2. Production errors are logged to a file and/or external logging service
3. Critical errors trigger alerts to the system administrator

6.4 User Feedback

Error messages displayed to users follow these guidelines:

1. Technical details are hidden from end-users to prevent security risks
2. Error messages are clear, concise, and actionable
3. When appropriate, users are guided on how to resolve the issue
4. For session timeouts or authentication issues, users are redirected to the login page

7 Configurable Parameters

The Hostel Management System uses a configuration management approach to allow customization without code changes. These parameters are stored in a central configuration file.

7.1 Application Configuration

Parameter Name	Definition and Usage	Dynamic?
PORT	The port number on which the application server runs	No
NODE_ENV	Environment setting (development, testing, production)	No
LOG_LEVEL	Level of logging detail (error, warn, info, debug)	Yes
SESSION_TIMEOUT	Session timeout period in minutes	Yes
MAX_LOGIN_ATTEMPTS	Maximum number of failed login attempts before account lockout	Yes
LOCKOUT_DURATION	Duration of account lockout in minutes after exceeding max login attempts	Yes

7.2 Database Configuration

| Parameter

7.2 Database Configuration

Parameter Name	Definition and Usage	Dynamic?
DB_HOST	Database server hostname or IP address	No
DB_PORT	Database server port number	No
DB_NAME	Name of the database	No
DB_USER	Database username for authentication	No
DB_PASSWORD	Database password for authentication	No
DB_POOL_SIZE	Maximum number of connections in the database connection pool	Yes
DB_IDLE_TIMEOUT	Timeout for idle database connections in milliseconds	Yes
DB_CONNECTION_TIMEOUT	Timeout for database connection attempts in milliseconds	Yes

7.3 Email Configuration

Parameter Name	Definition and Usage	Dynamic?
SMTP_HOST	SMTP server hostname for sending emails	No
SMTP_PORT	SMTP server port number	No
SMTP_USER	SMTP server username	No
SMTP_PASSWORD	SMTP server password	No
SMTP_FROM_EMAIL	Default 'from' email address for system-generated emails	Yes
SMTP_FROM_NAME	Default 'from' name for system-generated emails	Yes
EMAIL_TEMPLATE_PATH	Path to email templates directory	No

7.4 Payment Gateway Configuration

Parameter Name	Definition and Usage	Dynamic?
PAYMENT_GATEWAY_URL	URL of the payment gateway API	No
PAYMENT_GATEWAY_API_KEY	API key for the payment gateway	No
PAYMENT_GATEWAY_SECRET	Secret key for the payment gateway	No
PAYMENT_SUCCESS_URL	URL to redirect after successful payment	Yes
PAYMENT_FAILURE_URL	URL to redirect after failed payment	Yes
PAYMENT_CALLBACK_URL	URL for payment gateway callbacks	No
ALLOWED_PAYMENT_METHODS	Comma-separated list of enabled payment methods	Yes

7.5 Notification Configuration

Parameter Name	Definition and Usage	Dynamic?
SMS_GATEWAY_URL	URL of the SMS gateway API	No
SMS_GATEWAY_API_KEY	API key for the SMS gateway	No
SMS_SENDER_ID	Sender ID for SMS notifications	Yes
NOTIFICATION_BATCH_SIZE	Maximum number of notifications processed in a single batch	Yes
ENABLE_EMAIL_NOTIFICATIONS	Flag to enable/disable email notifications	Yes
ENABLE_SMS_NOTIFICATIONS	Flag to enable/disable SMS notifications	Yes
ENABLE_IN_APP_NOTIFICATIONS	Flag to enable/disable in-app notifications	Yes

7.6 Security Configuration

Parameter Name	Definition and Usage	Dynamic?
JWT_SECRET	Secret key for JWT token generation and validation	No
JWT_EXPIRY	JWT token expiry time in minutes	Yes
PASSWORD_SALT_ROUNDS	Number of salt rounds for password hashing	No
CORS_ALLOWED_ORIGINS	Comma-separated list of allowed origins for CORS	Yes
ENABLE_RATE_LIMITING	Flag to enable/disable rate limiting	Yes
RATE_LIMIT_WINDOW	Time window for rate limiting in milliseconds	Yes
RATE_LIMIT_MAX_REQUESTS	Maximum number of requests allowed in the time window	Yes

8 Quality of Service

8.1 Availability

The Hostel Management System is designed to provide high availability to ensure users can access critical hostel management functions at all times.

8.1.1 Availability Requirements

- The system must be available 24/7 with a target uptime of 99.9% (allowing for approximately 8.76 hours of downtime per year)
- Planned maintenance windows should be scheduled during low-usage periods, typically between 2 AM and 4 AM
- Critical functions like fee payment processing must maintain even higher availability (99.99%)

8.1.2 Availability Design Features

- **Database Redundancy**: Implement database replication to ensure data availability even in case of a primary database failure

- **Distributed Architecture**: Deploy the application across multiple servers to eliminate single points of failure
- **Load Balancing**: Distribute user traffic across multiple application instances
- **Health Monitoring**: Implement health checks and automatic recovery for application components
- **Graceful Degradation**: Design the system to continue functioning with reduced capabilities when non-critical components fail
- **Caching**: Implement strategic caching to reduce database load and improve availability during peak usage

8.1.3 Maintenance Strategy

- **Rolling Updates**: Implement rolling updates to avoid complete system downtime during deployments
- **Database Maintenance**: Schedule regular database maintenance during off-peak hours
- **Advance Notification**: Notify users of planned maintenance at least 48 hours in advance
- **Maintenance Mode**: Implement a maintenance mode that allows administrative functions while suspending regular user operations

8.2 Security and Authorization

Security is a critical aspect of the Hostel Management System, especially considering the sensitive nature of student and payment information.

8.2.1 Authentication Security

- **Password Policy**: Enforce strong password requirements (minimum 8 characters, combination of uppercase and lowercase letters, numbers, and special characters)
- **Password Hashing**: Use bcrypt for secure password hashing with appropriate salt rounds
- **Account Lockout**: Implement account lockout after multiple failed login attempts
- **Multi-factor Authentication**: Offer multi-factor authentication for administrative accounts
- **Session Management**: Implement secure session handling with appropriate timeouts and renewal mechanisms
- **JWT Security**: Use signed and encrypted JWTs with appropriate expiration times

8.2.2 Authorization Framework

- **Role-Based Access Control (RBAC)**: Implement RBAC with the following roles:
  - Student: Access to personal profile, room booking, fee payment, and requests
  - Parent: Access to child's profile, fee payment, and payment history

    - Warden: Access to room management, student management, request handling, and notifications
    - Administrator: Access to all system functions and configuration
- **Permission Management**: Define granular permissions that can be assigned to roles
- **Data Access Control**: Implement row-level security to ensure users can only access authorized data
- **API Security**: Secure all API endpoints with appropriate authentication and authorization checks

8.2.3 Data Security

- **Encryption**: Encrypt sensitive data at rest and in transit
- **Data Masking**: Implement data masking for sensitive information in logs and reports
- **Secure File Handling**: Ensure secure handling of uploaded documents and files
- **Input Validation**: Implement thorough input validation to prevent injection attacks
- **Output Encoding**: Properly encode all output to prevent XSS attacks
- **SQL Injection Prevention**: Use parameterized queries for all database operations

8.2.4 Audit and Compliance

- **Audit Logging**: Maintain comprehensive audit logs for all security-related events
- **User Activity Tracking**: Log all significant user actions for accountability
- **Compliance Monitoring**: Regularly review security logs and implement alerts for suspicious activities
- **Regular Security Reviews**: Conduct regular security reviews and penetration testing

8.3 Load and Performance Implications

The Hostel Management System is designed to handle varying loads with consistent performance.

8.3.1 Load Projections

- **Concurrent Users**: The system is expected to handle up to 500 concurrent users during peak periods (room booking season and fee payment deadlines)
- **Transaction Volume**:
    - Up to 1,000 room search operations per hour during peak periods
    - Up to 200 payment transactions per hour during fee payment deadlines
    - Up to 50 notification dispatches per minute
- **Data Growth**:
    - Approximately 5,000 new student records per year

- Approximately 10,000 new payment records per month
- Approximately 50,000 new notification records per month

8.3.2 Performance Requirements

- **Response Time**:
  - Page load time should be less than 2 seconds for 95% of requests
  - API response time should be less than 500ms for 95% of requests
  - Report generation should complete within 5 seconds for standard reports
- **Transaction Processing**:
  - Room booking transactions should complete within 3 seconds
  - Payment processing should complete within 5 seconds (excluding external payment gateway processing time)
- **Scalability**:
  - The system should maintain performance metrics when scaled to accommodate 50% increase in user load

8.3.3 Performance Design Features

- **Database Indexing**: Implement strategic indexing on frequently queried columns
- **Query Optimization**: Optimize SQL queries for performance
- **Connection Pooling**: Implement database connection pooling to reduce connection overhead
- **Caching Strategy**:
  - Cache room availability data to reduce database load during peak search periods
  - Cache static content and frequently accessed data
  - Implement Redis for distributed caching
- **Pagination**: Implement pagination for large data sets
- **Asynchronous Processing**:
  - Process notifications asynchronously
  - Generate reports asynchronously for large data sets
- **Resource Optimization**:
  - Optimize JavaScript, CSS, and HTML
  - Implement image optimization and lazy loading
  - Use content delivery networks (CDNs) for static assets

8.3.4 Database Performance Considerations

- **Table Partitioning**: Implement table partitioning for large tables (payments, notifications)

- **Archive Strategy**: Archive old data to maintain optimal performance of active data tables
- **Query Monitoring**: Implement query monitoring to identify and optimize slow queries
- **Database Scaling**: Design for both vertical and horizontal scaling of the database

8.4 Monitoring and Control

The Hostel Management System implements comprehensive monitoring and control mechanisms to ensure optimal performance and quick issue resolution.

8.4.1 Controllable Processes

- **Payment Processing**: Control the processing of payment transactions
- **Notification Dispatch**: Control the scheduling and processing of notifications
- **Report Generation**: Control the generation and distribution of system reports
- **Data Backup**: Control the scheduling and execution of data backups
- **User Session Management**: Control user session creation, validation, and termination

8.4.2 Monitoring Framework

- **Real-time Monitoring**: Implement real-time monitoring of key system metrics
  - CPU usage
  - Memory usage
  - Database connection pool status
  - Request response times
  - Error rates
  - User concurrency
- **Application Logging**: Implement structured logging for application events
  - Error logs
  - Transaction logs
  - Security logs
  - Performance logs
- **User Activity Monitoring**: Track and analyze user activity patterns
  - Page view statistics
  - Feature usage statistics
  - Session duration
  - Conversion rates (e.g., room search to booking)

8.4.3 Alerting System

- **Alert Thresholds**: Define thresholds for critical metrics that trigger alerts
  - Response time exceeding 2 seconds
  - Error rate exceeding 5%
  - CPU usage exceeding 80%
  - Memory usage exceeding 80%
  - Database connection pool utilization exceeding 80%
- **Alert Channels**: Implement multiple alert channels
  - Email alerts
  - SMS alerts
  - Dashboard notifications
  - Integration with monitoring services

8.4.4 Administrative Dashboard

- **System Health Dashboard**: Provide real-time visibility into system health
  - Server status
  - Database status
  - Application component status
  - Error trends
  - Performance metrics
- **Operational Controls**:
  - Start/stop application components
  - Enable/disable features
  - Update configuration parameters
  - Execute maintenance tasks
- **Reporting Tools**:
  - System usage reports
  - Performance reports
  - Error reports
  - Security reports

8.4.5 Incident Management

- **Incident Detection**: Automatically detect and log system incidents
- **Incident Tracking**: Maintain a log of detected incidents and their resolution
- **Root Cause Analysis**: Implement tools to support root cause analysis

- **Recovery Procedures**: Document and implement recovery procedures for common incidents

Appendix A: Technology Stack Details

A.1 Frontend Technology Stack

- **HTML5**: For structuring web pages
- **CSS3**: For styling web pages
- **JavaScript (ES6+)**: For client-side scripting
- **jQuery**: JavaScript library for DOM manipulation
- **Bootstrap**: CSS framework for responsive design
- **Chart.js**: For data visualization
- **Axios**: For HTTP requests
- **JWT-Decode**: For client-side JWT handling

A.2 Backend Technology Stack

- **Node.js**: JavaScript runtime environment
- **Express.js**: Web application framework
- **MySQL**: Relational database management system
- **Sequelize**: ORM for MySQL
- **Bcrypt**: For password hashing
- **JSON Web Token (JWT)**: For authentication
- **Multer**: For file uploads
- **Nodemailer**: For email sending
- **Winston**: For logging
- **Joi**: For input validation

A.3 Development Tools

- **Git**: For version control
- **Babel**: For JavaScript transpiling
- **Webpack**: For bundling assets
- **ESLint**: For code quality
- **Jest**: For unit testing
- **Postman**: For API testing
- **MySQL Workbench**: For database design and administration

Appendix B: API Documentation

B.1 Authentication API

B.1.1 Login

- **Endpoint**: POST /api/auth/login
- **Description**: Authenticates a user and returns a JWT token
- **Request Body**:

```json
{
  "email": "user@example.com",
  "password": "password123"
}
```

- **Response**:

```json
{
  "success": true,
  "message": "Login successful",
  "data": {
    "token": "jwt_token_here",
    "user": {
      "id": 1,
      "username": "johndoe",
      "email": "user@example.com",
      "role": "student"
    }
  }
}
```

B.1.2 Register

- **Endpoint**: POST /api/auth/register
- **Description**: Registers a new user
- **Request Body**:

```json
{
  "username": "johndoe",
  "email": "user@example.com",
  "password": "password123",
  "role": "student",
  "contactNumber": "1234567890"
}
```

- **Response**:

```json
{
  "success": true,
  "message": "Registration successful",
  "data": {
    "id": 1,
    "username": "johndoe",
    "email": "user@example.com",
    "role": "student"
  }
}
```

B.2 Room Management API

B.2.1 Get Available Rooms

- **Endpoint**: GET /api/rooms/available
- **Description**: Returns a list of available rooms based on criteria
- **Query Parameters**:
  - building (optional): Building ID
  - floor (optional): Floor number
  - type (optional): Room type
  - capacity (optional): Room capacity
- **Response**:

json


```
{
  "success": true,
  "message": "Available rooms retrieved",
  "data": {
    "rooms": [
      {
        "id": 1,
        "roomNumber": "A-101",
        "building": "Building A",
        "floor": 1,
        "type": "single",
        "capacity": 1,
        "monthlyFee": 5000,
        "amenities": ["AC", "Attached Bathroom"]
      }
    ],
    "totalCount": 1
  }
}
```


B.2.2 Book Room

- **Endpoint**: POST /api/rooms/book
- **Description**: Books a room for a student
- **Request Body**:

json


```
{
  "roomId": 1,
  "startDate": "2025-05-01"
}
```


- **Response**:

json


```
{
  "success": true,
  "message": "Room booked successfully",
  "data": {
    "bookingId": 1,
    "roomId": 1,
    "studentId": 1,
    "status": "pending",
    "invoiceId": 1
  }
}
```


B.3 Payment API

B.3.1 Get Invoices

- **Endpoint**: GET /api/payments/invoices
- **Description**: Returns a list of invoices for the authenticated user
- **Response**:

json


```
{
  "success": true,
  "message": "Invoices retrieved successfully",
  "data": {
    "invoices": [
      {
        "id": 1,
        "amount": 5000,
        "dueDate": "2025-05-15",
        "status": "pending",
        "items": [
          {
            "description": "Room fee for May 2025",
            "amount": 5000
          }
        ]
      }
    ],
    "totalCount": 1
  }
}
```

B.3.2 Process Payment

- **Endpoint**: POST /api/payments/process
- **Description**: Processes a payment for an invoice
- **Request Body**:

```json
{
  "invoiceId": 1,
  "paymentMethod": "credit_card",
  "cardNumber": "4111111111111111",
  "expiryMonth": "12",
  "expiryYear": "2025",
  "cvv": "123"
}
```

- **Response**:

```json
{
  "success": true,
  "message": "Payment processed successfully",
  "data": {
    "paymentId": 1,
    "transactionId": "tx_123456789",
    "amount": 5000,
    "status": "successful",
    "receiptUrl": "/api/payments/receipt/1"
  }
}
```

Appendix C: Database Schema SQL

sql

```sql
-- Create users table
CREATE TABLE users (
  user_id INT AUTO_INCREMENT PRIMARY KEY,
  username VARCHAR(50) NOT NULL UNIQUE,
  email VARCHAR(100) NOT NULL UNIQUE,
  password_hash VARCHAR(255) NOT NULL,
  role ENUM('student', 'parent', 'warden', 'admin') NOT NULL,
  contact_number VARCHAR(20) NOT NULL,
  profile_picture VARCHAR(255),
  status ENUM('active', 'inactive', 'suspended') NOT NULL DEFAULT 'active',
  created_at TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
  updated_at TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
);

-- Create students table
CREATE TABLE students (
  student_id INT AUTO_INCREMENT PRIMARY KEY,
  user_id INT NOT NULL,
  roll_number VARCHAR(20) NOT NULL UNIQUE,
  department VARCHAR(50) NOT NULL,
  year INT NOT NULL,
  parent_id INT,
  room_id INT,
  created_at TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
  updated_at TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
  FOREIGN KEY (user_id) REFERENCES users(user_id) ON DELETE CASCADE,
  FOREIGN KEY (parent_id) REFERENCES parents(parent_id) ON DELETE SET NULL,
  FOREIGN KEY (room_id) REFERENCES rooms(room_id) ON DELETE SET NULL
);

-- Create parents table
CREATE TABLE parents (
  parent_id INT AUTO_INCREMENT PRIMARY KEY,
  user_id INT NOT NULL,
  created_at TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
  updated_at TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
  FOREIGN KEY (user_id) REFERENCES users(user_id) ON DELETE CASCADE
);

-- Create buildings table
CREATE TABLE buildings (
  building_id INT AUTO_INCREMENT PRIMARY KEY,
  name VARCHAR(100) NOT NULL,
  description TEXT,
  floors INT NOT NULL,
  address TEXT NOT NULL,
```

```sql
  created_at TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
  updated_at TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
);

-- Create rooms table
CREATE TABLE rooms (
  room_id INT AUTO_INCREMENT PRIMARY KEY,
  room_number VARCHAR(20) NOT NULL,
  building_id INT NOT NULL,
  floor INT NOT NULL,
  capacity INT NOT NULL,
  type ENUM('single', 'double', 'shared') NOT NULL,
  description TEXT,
  status ENUM('available', 'reserved', 'occupied', 'maintenance') NOT NULL DEFAULT 'availa
  monthly_fee DECIMAL(10,2) NOT NULL,
  created_at TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
  updated_at TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
  FOREIGN KEY (building_id) REFERENCES buildings(building_id) ON DELETE CASCADE,
  UNIQUE KEY (building_id, room_number)
);

-- Additional tables would be created for room_amenities, bookings, allocations,
-- invoices, invoice_items, payments, notifications, notification_recipients, and request
```