DEPARTMENT RESOURCE BOOKING SYSTEM

Team: Laiba Ahamd (BSSE23085), Turab Hashmi(BSSE23002)

Submitted to: Dr. Zunnurain Hussain, Sir Umair Makhdoom

Institute: Information Technology University (ITU)

Date: 4 January 2026

Abstract

The efficient management of institutional resources such as classrooms, laboratories, and equipment is critical for the smooth operation of academic departments. Traditional manual methods of booking these resources often lead to conflicts, double bookings, and administrative inefficiencies. This project details the design and implementation of a "Department Resource Booking System," a full-stack web application developed to automate and streamline this process. The system supports two primary user roles: Admins, who manage resources and view all bookings, and Users (faculty/staff), who can view available resources and book them according to their needs. The application is built using React.js for the frontend and Node.js with Express.js for the backend, ensuring a responsive and dynamic user experience. Data persistence is managed using AWS RDS, and the application is deployed on the Amazon Web Services (AWS) cloud platform, specifically utilizing EC2 for the backend API and S3 for static frontend hosting. Security is enforced through JSON Web Token (JWT) authentication, AWS Identity and Access Management (IAM) roles, and email domain restrictions limited to `@itu.edu.pk`. This report provides a comprehensive overview of the system architecture, database design, implementation details, security measures, and deployment strategies, concluding with testing results and recommendations for future enhancements.

Acknowlegement

The Team express their sincere gratitude to the faculty and staff of the Information Technology University of Punjab for their continuous support and guidance during the course of this project. We are especially thankful to our supervisors, **Dr. Zunnurain** and **Umair Makhdom**, whose insightful suggestions, constructive feedback, and expertise in system architecture and AWS best practices greatly contributed to the successful completion of this work.

Table of Contents

List of Figures (LOF)

List of Tables (LOT)

Introduction

In academic institutions, the management of shared physical resources is a fundamental administrative task. Departments must allocate classrooms for lectures, laboratories for practical sessions, and specialized equipment for research or demonstrations. Historically, these tasks have been performed using manual logbooks, spreadsheets, or unstructured email chains. While these methods may function in very small environments, they become unscalable and error-prone as the number of users and resources increases.

The transition toward digital solutions offers significant advantages in terms of data integrity, accessibility, and operational efficiency. A web-based booking system allows users to check real-time availability and reserve resources without direct intervention from administrative staff.

This project, the "Department Resource Booking System," aims to provide a centralized digital platform for managing departmental assets. By leveraging modern web technologies and cloud infrastructure, the system ensures high availability and secure access for authorized personnel.

Problem Statement

The current manual resource management system faces several critical challenges:

1. Double Bookings: Without a centralized real-time system, two different users may book the same resource for the same time slot, leading to scheduling conflicts.

2. Lack of Visibility: Users cannot easily check the availability of resources without contacting administration, resulting in inefficient planning.

3. Administrative Burden: Staff are required to manually update spreadsheets and verify availability, consuming time that could be spent on more productive tasks.

4. Security and Tracking: Physical logbooks are susceptible to damage or loss, and tracking who modified a record is often impossible.

5. Accessibility: Manual records are accessible only during office hours, hindering the ability of faculty to plan resources in advance or outside of standard working hours.

Objectives

The primary objectives of this project are:

1. To develop a web-based application that allows authenticated users to view and book institutional resources.

2. To create an admin dashboard for the management of resources and the oversight of all bookings.

3. To implement a secure authentication mechanism using JWT and restrict access to users with a valid `@itu.edu.pk` email address.

4. To deploy the application on a scalable cloud infrastructure using AWS services (EC2, S3, RDS).

5. To ensure the system is responsive, user-friendly, and performs reliably under concurrent user load.

Scope of the System

The system is designed to manage resources within a specific academic department. The scope includes:

- User Roles: The system strictly distinguishes between "Admin" and "User."
- Resource Types: Classrooms, Labs, and Equipment.
- Booking Logic: Time-slot based bookings with conflict detection.
- Authentication: Email/password login restricted to the `itu.edu.pk` domain.

Out of Scope:

- Managing course schedules or semester timetables (this is for ad-hoc bookings only).
- Integration with external university calendars or email systems (beyond the initial domain restriction).
- Payment processing for resources.
- Multi-department support (the system is designed for a single department instance).

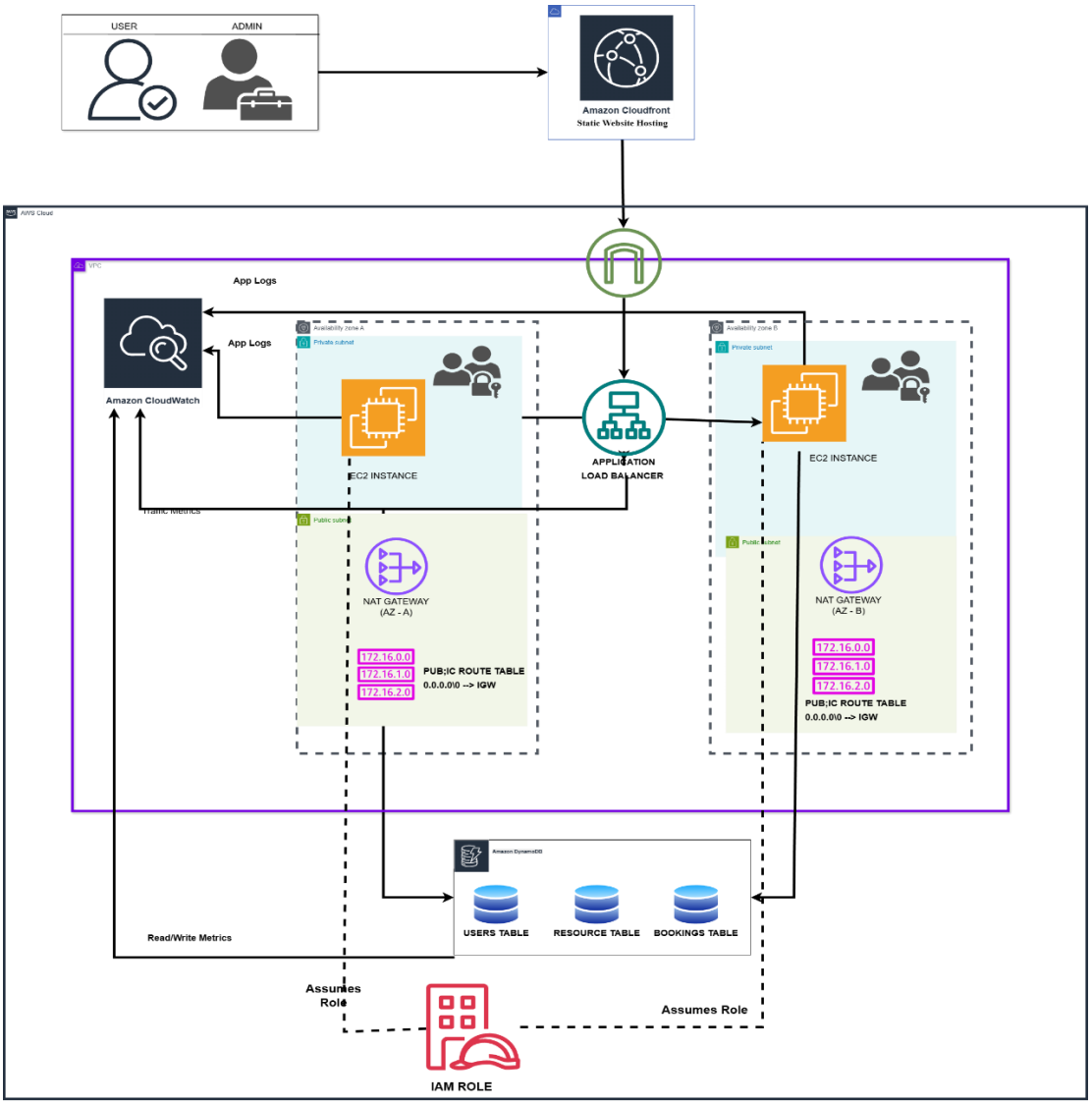

Figure 1: Architecture diagram

System Architecture

The Department Resource Booking System follows a client-server architecture, separated into distinct frontend and backend components. The system is hosted entirely on Amazon Web Services (AWS) to ensure scalability and reliability.

The Frontend is a Single Page Application (SPA) built with React.js. It is responsible for rendering the user interface, handling user inputs, and communicating with the backend server. The build artifacts of the React application are stored in AWS S3 and distributed via CloudFront for low-latency access.

The Backend is a RESTful API built with Node.js and Express.js. It handles business logic, authentication, and database interactions. The backend runs on an AWS EC2 instance within a secured Virtual Private Cloud (VPC).

The Database layer uses AWS RDS (Relational Database Service) running MySQL to store persistent data regarding users, resources, and bookings.

Communication between the frontend and backend occurs over HTTPS (HTTP Secure) using JSON data format.

*Figure 1: System Architecture Diagram illustrates the interaction between the Client (Browser), the S3 Bucket, the EC2 Instance (API), and the RDS Database.

Functional Requirements

Admin Features

- Resource Management: The Admin can create, read, update, and delete (CRUD) resources. This includes adding details such as resource name, type (Classroom, Lab, Equipment), capacity, and location.
- View All Bookings: The Admin has visibility into all bookings made by any user across the system.
- Cancel Bookings: The Admin has the authority to cancel any booking if necessary.

User Features

- Registration and Login: Users can register using an email and password. The system validates that the email belongs to the `@itu.edu.pk` domain.
- View Resources: Users can browse the list of available resources and filter them by type.
- Create Booking: Users can book a resource by specifying a start date/time, end date/time, and a purpose.
- View My Bookings: Users can view the history and status of their own specific bookings.
- Cancel Own Bookings: Users can cancel bookings they have created, provided the start time has not passed.

Non-Functional Requirements

- Security: The system must protect user data and prevent unauthorized access. Passwords must be hashed, and sensitive API endpoints must require a valid JWT. Access must be restricted to the university domain.
- Performance: The API should respond to requests within 200-500ms under normal load. Page loads should be optimized using code splitting and lazy loading in React.
- Scalability: The use of AWS services (EC2 and S3) allows the system to scale horizontally if the number of users increases significantly.
- Usability: The interface should be intuitive, requiring minimal training for new users. The design should be responsive to work on desktop and tablet screens.

Database Design

The database schema consists of three main entities: Users, Resources, and Bookings. The relationships are defined as follows: A User makes a Booking, and a Booking is made for a Resource.

Table 1: User Entity Attributes:

Table 1: User Entity Table

Attribute	Data Type	Description
user_id	INT (PK)	Unique identifier for the user
name	VARCHAR	Full name of the user
email	VARCHAR	User email (must be @itu.edu.pk)
password_hash	VARCHAR	Secured password hash
role	ENUM	Role of the user ('Admin' or 'User')

Table 2: Resource Entity Attributes

Table 2: Resource Entity Attributes

Attribute	Data Type	Description
resource_id	INT (PK)	Unique identifier for the resource
name	VARCHAR	Name of the classroom/lab/equipment
type	VARCHAR	Category (e.g., Classroom, Lab)
capacity	INT	Number of people the resource can hold
is_active	BOOLEAN	Status of the resource (Available/Inactive)

Table 3: Booking Entity Attributes

Table 3: Booking Entity Attributes

Attribute	Data Type	Description
booking_id	INT (PK)	| Unique identifier for the booking
user_id	INT (FK)	Reference to the User who made the booking
resource_id	INT (FK)	Reference to the booked Resource
start_time	DATETIME	Start date and time of the booking
end_time	DATETIME	End date and time of the booking
status	VARCHAR	Current status (Confirmed, Cancelled)

Frontend (React.js)

The frontend is structured into components: `Login`, `Dashboard`, `ResourceList`, and `BookingForm`. State management is handled using React Hooks (`useState`, `useEffect`). Axios is used as the HTTP client to send asynchronous requests to the backend API. The application is styled using CSS modules to ensure component-scoped styling and maintainability.

Backend (Node.js/Express)

The backend follows the Model-View-Controller (MVC) pattern. Routes are defined in `routes/`, controllers handle the logic in `controllers/`, and database interactions are abstracted into models using an ORM (like Sequelize) or a query builder.

Authentication Flow

When a user logs in, the backend verifies credentials. Upon success, it generates a signed JWT containing the user's ID and role. This token is sent back to the client. The client stores this token (typically in localStorage) and attaches it in the `Authorization` header (as `Bearer <token>`) for every subsequent request. The backend middleware verifies this token before granting access to protected routes.

Security Implementation

To ensure the integrity and security of the system, multiple layers of security have been implemented:

1. JSON Web Tokens (JWT): JWTs are used for stateless authentication. The tokens are signed using a strong secret key stored in environment variables on the server. Tokens have a defined expiration time to minimize the risk of session hijacking.

2. Email Domain Restriction: During registration and login, the backend validates the email string. If the email does not end with `@itu.edu.pk`, the request is rejected immediately, ensuring only authorized university personnel can access the system.

3. Password Hashing: User passwords are never stored in plain text. The `bcrypt` library is used to salt and hash passwords before storing them in the RDS database.

4. AWS Security Groups: Network security is managed via AWS Security Groups, which act as virtual firewalls. Only inbound traffic on specific ports (e.g., port 80 for HTTP, 443 for HTTPS, and a specific port for SSH access from a restricted IP) is allowed to the EC2 instance.

5. IAM Roles: The EC2 instance uses an IAM role to securely connect to the RDS database without hardcoding database credentials in the source code.

Deployment on AWS

The deployment strategy utilizes Infrastructure as a Service (IaaS) and Platform as a Service (PaaS) components from AWS.

IAM Setup

An Identity and Access Management (IAM) user was created with programmatic access to manage AWS resources. Additionally, an EC2 Instance Profile with an IAM role was attached to the server to grant it permission to write logs to CloudWatch.

EC2 Deployment

The backend application is deployed on an Amazon EC2 t2.micro instance running Ubuntu Linux.

- Node.js and NPM are installed on the instance.

- The code is pulled from a Git repository.
- Process Manager 2 (PM2) is used to keep the Node.js application running in the background and to restart it automatically if it crashes.

Database Configuration

An AWS RDS instance running MySQL is launched. It is placed in a private subnet within the VPC to prevent direct public internet access. The security group for the RDS instance is configured to accept traffic only from the security group of the EC2 instance.

CloudWatch Logging

PM2 is configured to stream application logs to AWS CloudWatch. This allows real-time monitoring of errors and API performance metrics, facilitating debugging and operational oversight.

Testing and Verification Methodology

To ensure the reliability and correct functioning of both the backend and frontend components, a combination of **manual and automated testing techniques** was employed. These methods were used to verify that both servers were correctly deployed, accessible over the internet, and functioning as intended.

Backend Testing (API Verification)

Backend testing was conducted in multiple stages to validate server availability, API functionality, and runtime stability:

- Process Monitoring:
  The pm2 list command was used to confirm that the backend service was running in an *Online* state. Memory utilization was also monitored (e.g., approximately 90 MB) to ensure stable resource consumption.
- Local Connectivity Testing:
  API availability was verified locally by executing the command
  curl http://localhost:5000/api/auth/login
  to ensure that the server was correctly accepting and responding to requests.
- Public Accessibility Testing:
  To validate external access, the backend API endpoint was accessed directly via the public IP address using a web browser (e.g., http://54.82.99.197:5000/api/resources). This confirmed that the AWS Security Group configuration permitted inbound traffic to the backend service.

- Log Analysis:
  Runtime logs were monitored using pm2 logs to identify and resolve errors such as ERR_UNKNOWN_FILE_EXTENSION, ensuring smooth execution of the backend application.

Frontend Testing (User Interface and Integration)

Frontend testing focused on deployment verification, backend connectivity, and client-side error detection:

- Deployment Verification:
  The pm2 list command was used to confirm that the frontend application was actively running on port 3000.
- Service Binding Validation:
  The backend API endpoint was updated in frontend/src/services/api.ts to reflect the correct public IP address (54.82.99.197), ensuring proper communication between the frontend and backend services.
- Browser Console Monitoring:
  The browser's developer console was used to detect and diagnose client-side issues such as net::ERR_CONNECTION_REFUSED, helping identify blocked or misconfigured network connections.

Integration and Security Testing

End-to-end testing was performed to validate system integration and security configurations:

- AWS Security Group Configuration:
  Inbound rules for both launch-wizard-1 and ITU-Security-Group were reviewed to ensure that ports **3000 (Frontend)** and **5000 (Backend)** were open and accessible.
- End-to-End Authentication Testing:
  User login was tested through the deployed application to verify the authentication flow, including credential validation, token generation, and JWT verification logic implemented in auth.ts.

Results and Discussion

The final implementation of the Department Resource Booking System meets all defined objectives.

- Functionality: The system successfully allows Admins to manage resources and Users to book them. The conflict detection logic prevents overlapping bookings for the same resource.
- Performance: Hosted on AWS, the application loads quickly. API response times remain low, providing a smooth user experience.

- Security: The implementation of JWT and email domain restriction successfully ensures that only users with `@itu.edu.pk` emails can register and access the system.
- User Interface: The React dashboard is clean and intuitive. *Figure 4: User Interface - Booking Dashboard* and *Figure 5: User Interface - Admin Resource Management* demonstrate the clean layout and ease of navigation.

Limitations

Despite the successful implementation, the system has certain limitations:

1. No Real-time Notifications: Users do not receive email or SMS notifications when a booking is confirmed or cancelled by an admin.

2. Basic Conflict Resolution: While the system prevents double bookings, it does not suggest alternative time slots to the user if a requested slot is unavailable.

3. Single Domain Restriction: The system is hard-coded to accept only `@itu.edu.pk` emails. Changing this would require a code modification and redeployment.

Future Enhancements

To improve the system further, the following features are proposed:

1. Email Notifications: Integration with AWS SES (Simple Email Service) to send booking confirmations and reminders.

2. Calendar View: Implementing a calendar view (using libraries like FullCalendar) to visualize resource availability more effectively.

3. Recurring Bookings: Adding functionality to book a resource for a recurring schedule (e.g., "Every Monday for 4 weeks").

4. Waitlist Feature: Allowing users to join a waitlist if a resource is already booked for their desired time.

Conclusion

The Department Resource Booking System successfully addresses the inefficiencies of manual resource management. By leveraging a React frontend and Node.js backend deployed on AWS, the project provides a secure, scalable, and user-friendly solution for academic institutions. The integration of role-based access control, JWT authentication, and email domain restrictions ensures that the system is secure and exclusive to the institution's members. This project demonstrates the

practical application of full-stack web development skills and cloud computing principles to solve real-world administrative problems.

References:

React.js Documentation, "React Hooks," React, 2025. [Online].

https://react.dev/reference/react

AWS, "What is Amazon EC2?," Amazon Web Services, 2025. [Online].
https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/concepts.html

Auth0, "JSON Web Token (JWT) Fundamentals," Auth0, 2025. [Online].

https://jwt.io/introduction

Node.js Foundation, "Node.js v18 Documentation," Node.js, 2025. [Online].

https://nodejs.org/en/docs

PM2, "PM2 - Production Process Manager," Keymetrics, 2025. [Online].

https://pm2.keymetrics.io/docs/usage/quick-start/

AWS, "IAM Roles for Amazon EC2," Amazon Web Services, 2025. [Online].

https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/iam-roles-for-amazon-ec2.html

Amazon Web Services, "AWS Academy Learner Lab Student Guide," AWS Academy, 2025. [Online]. Available:

https://awsacademy.instructure.com

GITHUB LINK:

AWS_project_SDC