BRAC UNIVERSITY

Inspiring Excellence

CSE470: Software Engineering
Project Report
Project Title: One Stop Portal

Group No: 02, CSE470 Section: 08, Spring2025	
ID	**Name**
23341115	Yasin Rahman
21141014	Stanley Matthew Das
21301039	Arif Jawad Alvi

Submission Date: 21/05/2025

Table of Contents

1. Functional Requirements

Admins can view, add, update and delete users and can create, update and delete courses. They also have the capability of viewing and managing field bookings such as resolving conflicts and cancelling bookings. They also can generate reports of data. Admins can send notifications to users about any change and set permissions levels for staff, faculty, STs, etc.

Users can register and log in and based on the email domain, will be assigned a permission level after which, if they wish, they can do profile customisation, such as, uploading profile pictures, etc. They may request something to the University Board via a form. A payment portal from which users can connect to the university payment portal and pay in installments or in full is also available to them. They can interact with the university library, transportation and club activities, etc. Furthermore, users can communicate with each other through in-app messaging.

The system can display a list of all available courses with details, where users can search and filter courses and can enroll into a course but the system will prevent enrollment if the user is currently enrolled into it. The users can view the course materials and interact with popquiz, etc. upon completion (displaying the progress as a percentage bar) the system will display the gradesheet of the course and ask for a course and faculty review.

The system will show all slots available and users can filter using their preferred time as well as scheduled matches so users can book a versus match. If users already have booked a slot before, they can view their previous bookings and do one-click rebook. Once the user books, the system will send a confirmation message with booking details (date and time) to the user using in-app messaging for which the system will allow cancellations before 24 hours from the booked match time. A cancellation notification will be sent to the users who booked using in-app messaging. The system will send notifications using in-app messaging to alert users about booked slot utilising a time threshold set by the user.

2. Technology (Framework, Languages)

Language: JavaScript
Techstack: MERN (MongoDB, Express, React, Node)

3. Backend Development

Booking Model-This model handles reservations for facilities like "Field A", "Field B", or "Auditorium". Bookings include a user, facility, time range, status (e.g., pending, approved), and a flag to mark conflict resolution. The controller provides endpoints to view all bookings, cancel a booking, or mark a conflict as resolved, enabling smooth scheduling operations.

```
const mongoose = require('mongoose');

const BookingSchema = new mongoose.Schema({
  user: { type: mongoose.Schema.Types.ObjectId, ref: 'User',
required: true },
  facility: {
    type: String,
    required: true,
    enum: ['Field A', 'Field B', 'Auditorium']
  },
  startTime: { type: Date, required: true },
  endTime: { type: Date, required: true },
  status: {
    type: String,
    enum: ['pending', 'approved', 'rejected', 'cancelled'],
    default: 'pending'
  },
  conflictResolved: { type: Boolean, default: false }
}, { timestamps: true });

module.exports = mongoose.model('Booking', BookingSchema);
```

Booking Controller-

```
const Booking = require('../models/Booking');
```

```

// Get all bookings
exports.getBookings = async (req, res) => {
  try {
    const bookings = await Booking.find()
      .populate('user', 'name email') // Assuming User model
contains 'name' and 'email'
      .exec();
    res.json(bookings);
  } catch (err) {
    res.status(500).json({ error: 'Failed to fetch bookings' });
  }
};

// Cancel a booking (update status to cancelled)
exports.cancelBooking = async (req, res) => {
  try {
    const booking = await Booking.findByIdAndUpdate(
      req.params.id,
      { status: 'cancelled' }, // Change status to 'cancelled'
      { new: true }
    );
    if (!booking) {
      return res.status(404).json({ error: 'Booking not found'
});
    }
    res.json({ message: 'Booking cancelled successfully', booking
});
  } catch (err) {
    res.status(500).json({ error: err.message });
  }
};

// Resolve a booking conflict (set conflictResolved to true)
exports.resolveBookingConflict = async (req, res) => {
  try {
    const booking = await Booking.findById(req.params.id);
    if (!booking) {
      return res.status(404).json({ error: 'Booking not found'
});
    }

    // Resolve the conflict
    booking.conflictResolved = true;
    await booking.save();

    res.json({ message: 'Booking conflict resolved', booking });
  } catch (err) {
    res.status(500).json({ error: err.message });
  }
};
```

Course Model-The Course model represents academic courses with attributes like title, code, description, credits, assigned faculty, status, and an array of materials (PDFs, videos, or links). The controller supports creating, updating, fetching, and deleting courses. It ensures course codes are unique and links faculty to each course using a reference to the User model.

```
const mongoose = require('mongoose');
const materialSchema = new mongoose.Schema({
  title: { type: String, required: true },
  filename: { type: String, required: true },
    type: { type: String, enum: ['pdf', 'video', 'link'],
default: 'pdf', required: true }
  });




const CourseSchema = new mongoose.Schema({
  title: { type: String, required: true },
  code: { type: String, required: true, unique: true },
  description: String,
  credits: { type: Number, default: 3 },
  faculty: { type: mongoose.Schema.Types.ObjectId, ref: 'User' },
  status: { type: String, enum: ['active', 'archived'], default:
'active' },
  materials: [
    {
      title: { type: String, required: true },
      link: { type: String, required: true },
      type: { type: String, enum: ['pdf', 'video', 'link'],
default: 'pdf' }
    }
  ]
}, { timestamps: true });

module.exports = mongoose.model('Course', CourseSchema);
```

Course Controller

```
const mongoose = require('mongoose'); // Ensure mongoose is
required
const Course = require('../models/Course');

exports.createCourse = async (req, res) => {
  try {
    const { title, code, description, credits, faculty } =
req.body;

    // Ensure faculty is a valid ObjectId
    const validFacultyId = new mongoose.Types.ObjectId(faculty);
// Corrected line

    // Check if course code already exists
    const existingCourse = await Course.findOne({ code });
    if (existingCourse) {
      return res.status(400).json({ error: 'Course code already
exists' });
    }

    const course = new Course({ title, code, description,
credits, faculty: validFacultyId, materials });
    await course.save();
    res.status(201).json(course);
  } catch (err) {
    res.status(400).json({ error: err.message });
  }
};

exports.getCourses = async (req, res) => {
  try {
    const courses = await Course.find().populate('faculty', 'name
email');
    res.json(courses);
  } catch (err) {
    res.status(500).json({ error: 'Failed to fetch courses' });
  }
};

exports.updateCourse = async (req, res) => {
  try {
    // Ensure the faculty field is valid if it's being updated
    if (req.body.faculty) {
      req.body.faculty = new
mongoose.Types.ObjectId(req.body.faculty);
    }

    const course = await Course.findByIdAndUpdate(req.params.id,
req.body, { new: true });
    if (!course) {
      return res.status(404).json({ error: 'Course not found' });
    }
    res.json(course);
  } catch (err) {
    res.status(400).json({ error: err.message });
  }
};

exports.deleteCourse = async (req, res) => {
  try {
    const course = await Course.findByIdAndDelete(req.params.id);
    if (!course) {
      return res.status(404).json({ error: 'Course not found' });
    }
    res.json({ message: 'Course deleted' });
  } catch (err) {
    res.status(500).json({ error: err.message });
  }
};

exports.getCourseById = async (req, res) => {
  try {
    const course = await
Course.findById(req.params.id).populate('faculty', 'name email');
    if (!course) {
      return res.status(404).json({ error: 'Course not found' });
    }
    res.json(course);
  } catch (err) {
    res.status(500).json({ error: 'Failed to fetch course' });
  }
};
```

User Controller-The User model defines users with fields such as name, email, password, role (admin, faculty, or student), and status. It includes password hashing with bcrypt for secure storage. The controller handles user management—listing, adding, updating (including profile picture uploads), and deleting users. It also allows authenticated users to view and update their own profile with proper validations.

```
const User = require('../models/User');
const fs = require('fs');
const path = require('path');

module.exports = {
  // Get all users (admin access only)
  getUsers: async (req, res) => {
    try {
      const users = await User.find();
      res.json(users);
    } catch (err) {
      res.status(500).json({ error: err.message });
    }
  },

  // Add a new user (admin access)
  addUser: async (req, res) => {
    try {
      const user = new User(req.body);
      await user.save();
      res.status(201).json(user);
    } catch (err) {
      res.status(400).json({ error: err.message });
    }
  },

  // Update a user (admin or own profile)
  updateUser: async (req, res) => {
    try {
      const userId = req.user.userId; // Get the userId from the token
      // Check if the user is an admin or trying to update their own profile
      if (req.params.id !== userId && req.user.role !== 'admin') {
        return res.status(403).json({ error: 'You can only update your own profile or you need admin access' });
      }

      const user = await User.findByIdAndUpdate(req.params.id, req.body, { new: true });
      if (!user) {
        return res.status(404).json({ error: 'User not found' });
      }
      res.json(user);
    } catch (err) {
      res.status(400).json({ error: err.message });
    }
  },
  // Delete a user (admin access)
  deleteUser: async (req, res) => {
    try {
      const user = await User.findByIdAndDelete(req.params.id);
      if (!user) {
        return res.status(404).json({ error: 'User not found' });
      }
      res.json({ message: 'User deleted' });
    } catch (err) {
      res.status(500).json({ error: err.message });
    }
  },

  // View own profile (only accessible by the authenticated user)
  viewUserProfile: async (req, res) => {
    try {
      const user = await User
        .findById(req.user.userId)
        .select('-password');
      if (!user) return res.status(404).json({ error: 'User not
found' });

      // Ensure profilePictureUrl is properly formatted for
frontend
      if (user.profilePictureUrl) {
        // Make sure the URL is properly formatted - ensure
consistency
        user.profilePictureUrl =
user.profilePictureUrl.startsWith('http')
          ? user.profilePictureUrl
          :
`${req.protocol}://${req.get('host')}${user.profilePictureUrl}`;
      }

      res.json(user);
    } catch (err) {
      console.error('viewUserProfile error:', err);
      res.status(500).json({ error: 'Failed to fetch user
profile' });
    }
  },

  // Update own profile (only accessible by authenticated user)
  updateUserProfile: async (req, res) => {
    try {
      const user = await User.findById(req.user.userId);
      if (!user) return res.status(404).json({ error: 'User not
found' });

      // Update allowed fields
      if (req.body.name) {
        // Validate name
        if (req.body.name.length < 2 || req.body.name.length >
50) {
          return res.status(400).json({ error: 'Name must be
between 2 and 50 characters' });
        }
        user.name = req.body.name;
      }

      // Handle uploaded picture
      if (req.file) {
        // Validate file
        const allowedTypes = ['image/jpeg', 'image/png',
'image/gif'];
        const maxSize = 5 * 1024 * 1024; // 5MB

        if (!allowedTypes.includes(req.file.mimetype)) {
          // Remove uploaded file if invalid type
          fs.unlinkSync(req.file.path);
          return res.status(400).json({ error: 'Invalid file
type. Only JPEG, PNG and GIF are allowed.' });
        }

        if (req.file.size > maxSize) {
          // Remove uploaded file if too large
          fs.unlinkSync(req.file.path);
          return res.status(400).json({ error: 'File too large.
Maximum size is 5MB.' });
        }

        // If user already has a profile picture, delete the old
one
        if (user.profilePictureUrl) {
          try {
            const oldFilePath = user.profilePictureUrl.replace(
              `${req.protocol}://${req.get('host')}`, ''
            );
            const fullPath = path.join(__dirname, '..', 'public',
oldFilePath);
            if (fs.existsSync(fullPath)) {
              fs.unlinkSync(fullPath);
            }
          } catch (deleteErr) {
            console.error('Error deleting old profile picture:',
deleteErr);
            // Continue with the update even if deletion fails
          }
        }

        // Save a URL that your frontend can fetch
        const uploadPath =
`/uploads/profile/${req.file.filename}`;
        user.profilePictureUrl =
`${req.protocol}://${req.get('host')}${uploadPath}`;
      }

      const updated = await user.save();
      const { password, ...userData } = updated.toObject();
      res.json(userData);

    } catch (err) {
      console.error('updateUserProfile error:', err);
      res.status(500).json({ error: 'Failed to update profile'
});
    }
  },
};
```

User Model

```
const mongoose = require('mongoose');
const bcrypt = require('bcryptjs');

const UserSchema = new mongoose.Schema({
  name: { type: String, required: true },
  email: { type: String, required: true, unique: true },
  password: { type: String, required: true },
  role: {
    type: String,
    enum: ['admin', 'faculty', 'student'],
    default: 'student'
  },
  status: { type: String, enum: ['active', 'inactive'], default:
'active' }
}, { timestamps: true });

UserSchema.pre('save', async function(next) {
  if (!this.isModified('password')) return next();
  this.password = await bcrypt.hash(this.password, 10);
  next();
});

UserSchema.methods.comparePassword = async
function(candidatePassword) {
  return await bcrypt.compare(candidatePassword, this.password);
};

module.exports = mongoose.model('User', UserSchema);
```

Message Model-Messages are stored with sender, recipient, body, and a read flag. The controller allows users to send messages, fetch their conversation history, and mark specific messages as read. It ensures secure communication by linking messages to user accounts and verifying message ownership during operations.

```
const mongoose = require('mongoose');
```

```

const MessageSchema = new mongoose.Schema({
  sender:    { type: mongoose.Schema.Types.ObjectId, ref: 'User',
required: true },
  recipient: { type: mongoose.Schema.Types.ObjectId, ref: 'User',
required: true },
  body:      { type: String, required: true },
  read:      { type: Boolean, default: false }
}, { timestamps: true });

module.exports = mongoose.model('Message', MessageSchema);
```

Message Controller

```
// backend/controllers/messageController.js
const svc = require('../services/messageService');

exports.sendMessage = async (req, res) => {
  try {
    const { recipient, body } = req.body;
    if (!recipient || !body) {
      return res.status(400).json({ error: 'recipient and body
are required' });
    }
    const msg = await svc.createMessage({
      sender: req.user.id,
      recipient,
      body
    });
    return res.status(201).json(msg);
  } catch (err) {
    console.error(err);
    return res.status(500).json({ error: 'Failed to send message'
});
  }
};

exports.getConversations = async (req, res) => {
  try {
    const msgs = await svc.getUserMessages(req.user.id);
    return res.json(msgs);
  } catch (err) {
    console.error(err);
    return res.status(500).json({ error: 'Failed to load
messages' });
  }
};

exports.markRead = async (req, res) => {
  try {
    const updated = await svc.markMessageRead(req.params.id,
req.user.id);
    return res.json(updated);
  } catch (err) {
    if (err.message === 'NotFound') return res.status(404).json({
error: 'Message not found' });
    if (err.code === 'FORBIDDEN') return res.status(403).json({
error: 'Not your message' });
    console.error(err);
    return res.status(500).json({ error: 'Failed to mark read'
});
```

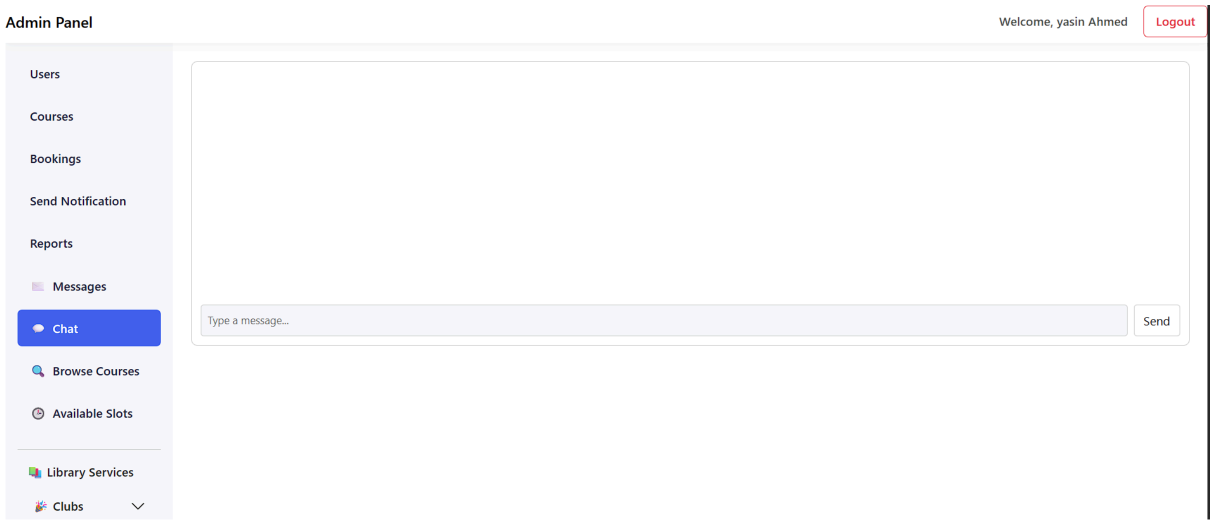
```
  }
};
```

Enrollment-This code manages course enrollments for students, allowing them to enroll in courses, view their enrollments, update their progress, and retrieve completed courses (gradesheet). It ensures only students can enroll and prevents duplicate enrollments. The `Enrollment` schema tracks user, course, enrollment date, progress percentage, and grade.

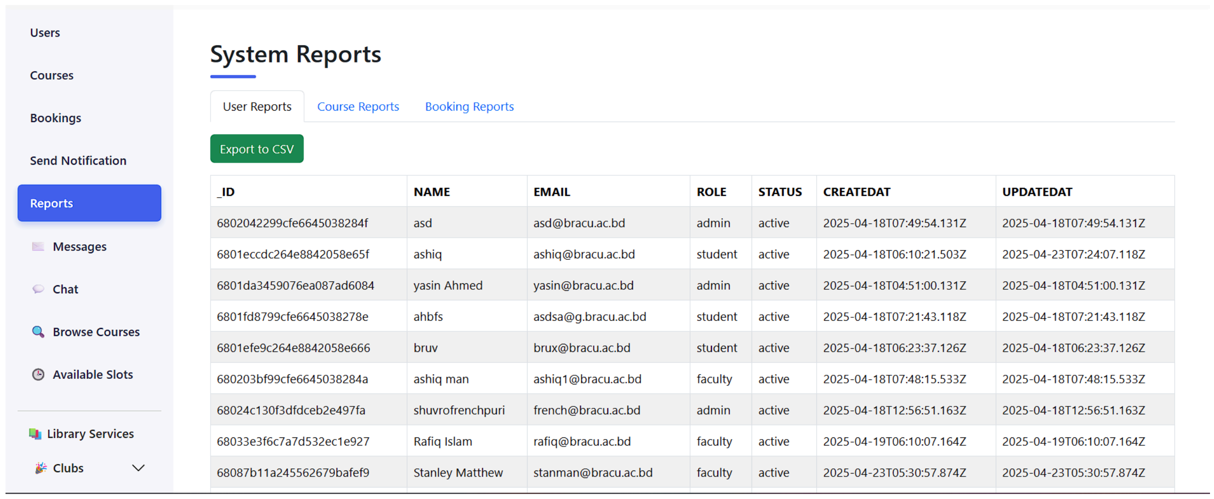
```
const mongoose = require('mongoose');
const enrollmentSchema = new mongoose.Schema({
  user: { type: mongoose.Schema.Types.ObjectId, ref: 'User',
required: true },
  course: { type: mongoose.Schema.Types.ObjectId, ref: 'Course',
required: true },
  enrolledAt: { type: Date, default: Date.now },
  progress: { type: Number, default: 0 }, // 0-100
  grade: { type: String, default: '' }
});
module.exports = mongoose.model('Enrollment', enrollmentSchema);
const Enrollment = require('../models/Enrollment');

exports.createEnrollment = async (req, res) => {
    if (req.user.role !== 'student') {
        return res.status(403).json({ msg: 'Access denied' });
    }
  const { course } = req.body;
  try {
    // prevent duplicate
    const exists = await Enrollment.findOne({ user: req.user.id,
course });
    if (exists) return res.status(400).json({ msg: 'Already
enrolled' });

    const newEnroll = new Enrollment({ user: req.user.id, course
});
    await newEnroll.save();
    res.status(201).json(newEnroll);
  } catch (err) {
    res.status(500).json({ error: err.message });
  }
};

exports.getMyEnrollments = async (req, res) => {
  try {
    const list = await Enrollment.find({ user: req.user.id
}).populate('course');
    res.json(list);
  } catch (err) {
    res.status(500).json({ error: err.message });
  }
};

exports.updateProgress = async (req, res) => {
  try {
    const updated = await Enrollment.findByIdAndUpdate(
      req.params.id,
      { progress: req.body.progress },
      { new: true }
    );
    res.json(updated);
  } catch (err) {
    res.status(500).json({ error: err.message });
  }
};

exports.getGradesheet = async (req, res) => {
  try {
    const completed = await Enrollment.find({ user: req.user.id,
progress: 100 });
    res.json(completed);
  } catch (err) {
    res.status(500).json({ error: err.message });
  }
};
```

4. User Interface Design

Chatbox



Report



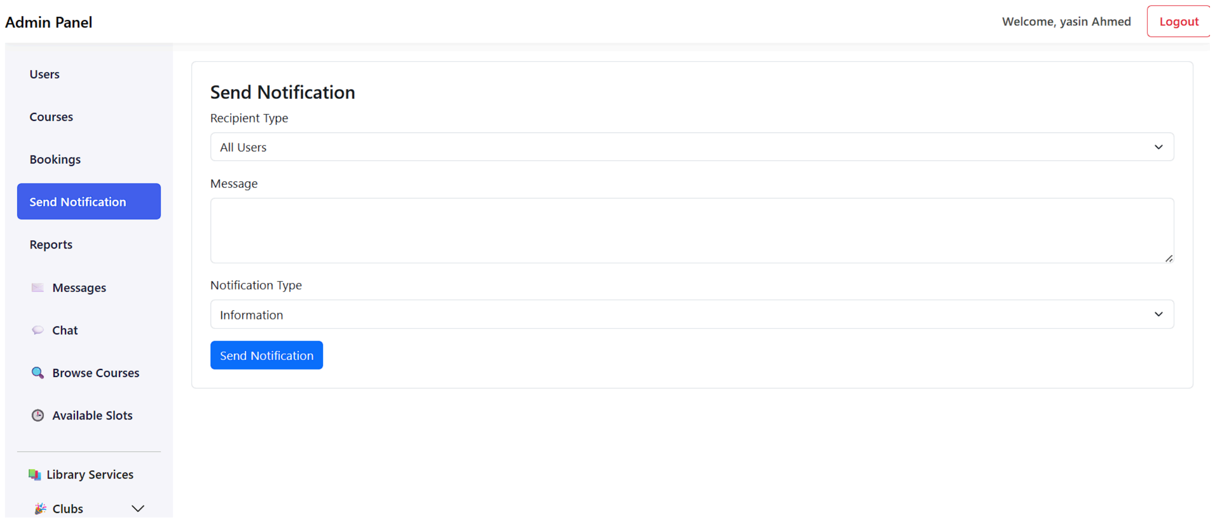
System Reports

User Reports | Course Reports | Booking Reports

Export to CSV

_ID	NAME	EMAIL	ROLE	STATUS	CREATEDAT	UPDATEDAT
6802042299cfe6645038284f	asd	asd@bracu.ac.bd	admin	active	2025-04-18T07:49:54.131Z	2025-04-18T07:49:54.131Z
6801eccdc264e8842058e65f	ashiq	ashiq@bracu.ac.bd	student	active	2025-04-18T06:10:21.503Z	2025-04-23T07:24:07.118Z
6801da3459076ea087ad6084	yasin Ahmed	yasin@bracu.ac.bd	admin	active	2025-04-18T04:51:00.131Z	2025-04-18T04:51:00.131Z
6801fd8799cfe6645038278e	ahbfs	asdsa@g.bracu.ac.bd	student	active	2025-04-18T07:21:43.118Z	2025-04-18T07:21:43.118Z
6801efe9c264e8842058e666	bruv	brux@bracu.ac.bd	student	active	2025-04-18T06:23:37.126Z	2025-04-18T06:23:37.126Z
680203bf99cfe6645038284a	ashiq man	ashiq1@bracu.ac.bd	faculty	active	2025-04-18T07:48:15.533Z	2025-04-18T07:48:15.533Z
68024c130f3dfdceb2e497fa	shuvrofrenchpuri	french@bracu.ac.bd	admin	active	2025-04-18T12:56:51.163Z	2025-04-18T12:56:51.163Z
68033e3f6c7a7d532ec1e927	Rafiq Islam	rafiq@bracu.ac.bd	faculty	active	2025-04-19T06:10:07.164Z	2025-04-19T06:10:07.164Z
68087b11a245562679bafef9	Stanley Matthew	stanman@bracu.ac.bd	faculty	active	2025-04-23T05:30:57.874Z	2025-04-23T05:30:57.874Z

Send Notification



Send Notification

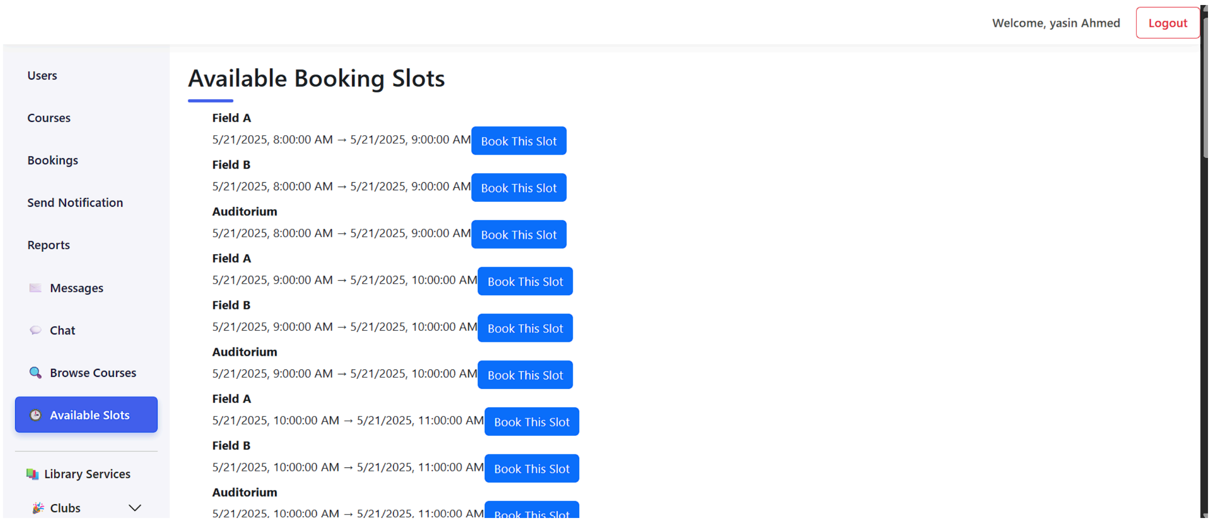
Recipient Type: All Users

Message:

Notification Type: Information

Send Notification

Available Slots


Welcome, yasin Ahmed

Logout

Users

Courses

Bookings

Send Notification

Reports

Messages

Chat

Browse Courses

Available Slots

Library Services

Clubs

Available Booking Slots

Field A
5/21/2025, 8:00:00 AM → 5/21/2025, 9:00:00 AM Book This Slot

Field B
5/21/2025, 8:00:00 AM → 5/21/2025, 9:00:00 AM Book This Slot

Auditorium
5/21/2025, 8:00:00 AM → 5/21/2025, 9:00:00 AM Book This Slot

Field A
5/21/2025, 9:00:00 AM → 5/21/2025, 10:00:00 AM Book This Slot

Field B
5/21/2025, 9:00:00 AM → 5/21/2025, 10:00:00 AM Book This Slot

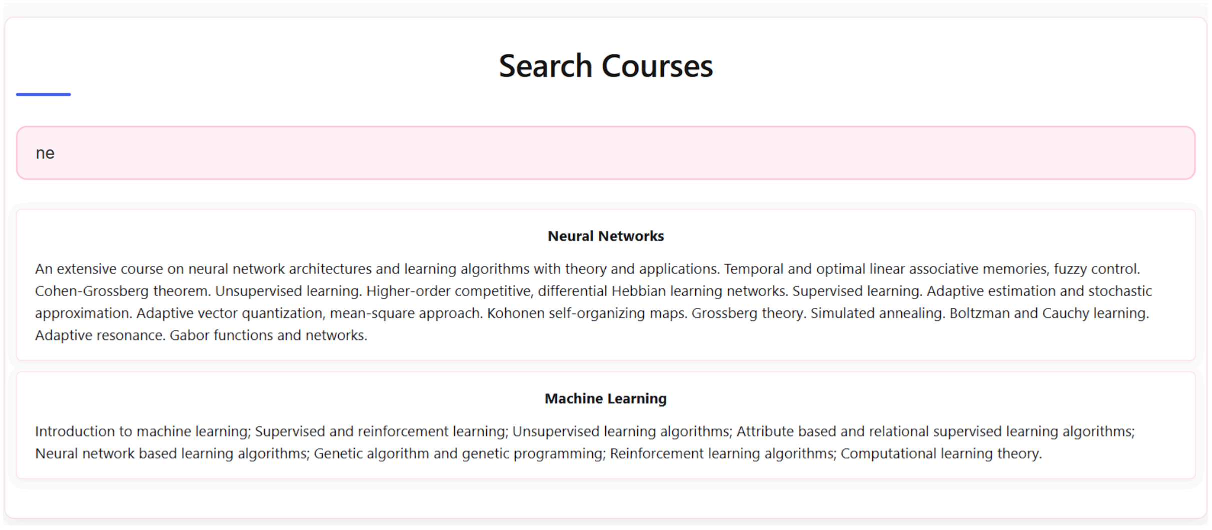
Auditorium
5/21/2025, 9:00:00 AM → 5/21/2025, 10:00:00 AM Book This Slot

Field A
5/21/2025, 10:00:00 AM → 5/21/2025, 11:00:00 AM Book This Slot

Field B
5/21/2025, 10:00:00 AM → 5/21/2025, 11:00:00 AM Book This Slot

Auditorium
5/21/2025, 10:00:00 AM → 5/21/2025, 11:00:00 AM Book This Slot

Search Courses


Search Courses

ne

Neural Networks

An extensive course on neural network architectures and learning algorithms with theory and applications. Temporal and optimal linear associative memories, fuzzy control. Cohen-Grossberg theorem. Unsupervised learning. Higher-order competitive, differential Hebbian learning networks. Supervised learning. Adaptive estimation and stochastic approximation. Adaptive vector quantization, mean-square approach. Kohonen self-organizing maps. Grossberg theory. Simulated annealing. Boltzman and Cauchy learning. Adaptive resonance. Gabor functions and networks.

Machine Learning

Introduction to machine learning; Supervised and reinforcement learning; Unsupervised learning algorithms; Attribute based and relational supervised learning algorithms; Neural network based learning algorithms; Genetic algorithm and genetic programming; Reinforcement learning algorithms; Computational learning theory.

5. Frontend Development

Course Search-CourseSearch component provides a search input for users to find courses by title. It debounces user input, then fetches matching courses from an API using Axios, showing loading status and results

```
import React, { useState, useEffect } from 'react';
import axios from 'axios';

// Import the CSS file
import './CourseSearch.css';

const CourseSearch = () => {
    const [search, setSearch] = useState('');
    const [courses, setCourses] = useState([]);
    const [loading, setLoading] = useState(false);
    const fetchCourses = async () => {
        if (search.trim() === '') {
            setCourses([]); // Clear courses if no search term
            return;
        }

        setLoading(true);
        try {
            const res = await axios.get('/api/courses', {
                params: { search },
                withCredentials: true,
            });
            setCourses(res.data);
        } catch (err) {
            console.error('Error fetching courses:', err);
        }
        setLoading(false);
    };

    useEffect(() => {
        // Only fetch courses when search input changes, and debounce
the requests
        const delayDebounceFn = setTimeout(() => {
            fetchCourses();
        }, 500); // 500ms delay after the user stops typing

        return () => clearTimeout(delayDebounceFn); // Cleanup the
previous timeout if input changes
    }, [search]); // Only call fetchCourses when search input changes

    return (
        <div className="container">
            <h2 className="heading">Search Courses</h2>
            <div className="search-container">
                <input
                    type="text"
                    placeholder="Search by course title"
                    value={search}
                    onChange={(e) => setSearch(e.target.value)}
                    className="search-input"
                />
            </div>
            {loading ? (
                <p className="loading">Loading...</p>
            ) : courses.length === 0 && search.trim() !== '' ? (
                <p className="no-results">No courses found.</p>
            ) : (
                <ul className="course-list">
                    {courses.map((course) => (
                        <li key={course._id} className="course-item">

<strong>{course.title}</strong>{course.description || 'No description
available'}
                            {course.materials &&
course.materials.length > 0 && (
                                <ul className="material-list">
                                    {course.materials.map((material,
index) => (
                                        <li key={index}
className="material-item">
                                            <span
className="material-type">{material.type.toUpperCase()}:</span>{' '}
                                            <a href={material.link}
target="_blank" rel="noopener noreferrer">{material.title}</a>
                                        </li>
                                    ))}
                                </ul>
                            )}
                        </li>
                    ))}
                </ul>

            )}
        </div>
    );
};

export default CourseSearch;
```

```
/* Soft container with a pastel background and gentle shadow */
.container {
    max-width: 900px;
    margin: auto;
    padding: 2.5rem;
```

```css
    background: #fefefe;
    border-radius: 20px;
    box-shadow: 0 12px 36px rgba(255, 182, 193, 0.2);
    transition: transform 0.3s ease, box-shadow 0.3s ease;
    border: 1px solid #f9dfe2;
}

/* Hover animation for container */
.container:hover {
    transform: translateY(-6px);
    box-shadow: 0 18px 48px rgba(255, 182, 193, 0.3);
}

.heading {
    text-align: center;
    font-size: 2.2rem;
    font-weight: 600;
    color: #1a1a1a;
    margin-bottom: 2rem;
    letter-spacing: -0.5px;
}


/* Search input centered */
.search-container {
    display: flex;
    justify-content: center;
    margin-bottom: 2rem;
}

/* Search input field */
.search-input {
    width: 100%;
    padding: 14px 20px;
    font-size: 1.15rem;
    border-radius: 12px;
    border: 2px solid #ffc8dd;
    background: #fff0f6;
    color: #444;
    font-family: 'Poppins', sans-serif;
    transition: all 0.3s ease;
}
```

```css
/* Focused input style */
.search-input:focus {
    border-color: #ffafcc;
    box-shadow: 0 0 10px #ffc8dd80;
    outline: none;
}

/* Loading and empty message styles */
.loading,
.no-results {
    text-align: center;
    font-size: 1.3rem;
    color: #aaa;
    font-family: 'Poppins', sans-serif;
    margin-top: 1rem;
}

/* Course list styles */
.course-list {
    list-style: none;
    padding: 0;
    margin: 0;
}

/* Individual course card */
.course-item {
    margin-bottom: 20px;
    padding: 24px;
    background: #fff5f8;
    border-radius: 16px;
    border: 1px solid #ffe2e9;
    box-shadow: 0 6px 20px rgba(255, 182, 193, 0.1);
    transition: transform 0.3s ease, background-color 0.3s ease;
    display: flex;
    flex-direction: column;
    gap: 12px;
}

/* Hover effect */
.course-item:hover {
    transform: translateY(-4px);
    background-color: #ffe8f1;
}
```

```
/* Title styling */
.course-item .title {
    font-size: 1.6rem;
    font-weight: 600;
    color: #e63946;
    font-family: 'Baloo 2', cursive;
    letter-spacing: 0.5px;
}

/* Description styling */
.course-item .description {
    font-size: 1.05rem;
    color: #555;
    line-height: 1.6;
    font-family: 'Poppins', sans-serif;
    padding-left: 2px;
}
```

Chatbox-ChatBox component manages real-time messaging between the current user and a selected recipient. It fetches past messages via API, listens for new incoming messages through WebSocket, and displays them with auto-scrolling. Users can type and send messages, which are transmitted over WebSocket. The component also handles socket setup and cleanup on mount/unmount or recipient changes.

```
// frontend/src/components/ChatBox.js
import React, { useEffect, useState, useRef } from 'react';
import {
  setupSocket,
  subscribeToMessages,
  subscribeToMessageSent,
  sendMessageWS,
  disconnectSocket,
} from '../services/socket';
import api from '../services/api';

const ChatBox = ({ recipient, currentUserId }) => {
  const [messages, setMessages] = useState([]);
  const [input, setInput] = useState('');
  const bottomRef = useRef(null);

  // Scroll to the bottom whenever messages change
  const scrollToBottom = () => {
    bottomRef.current?.scrollIntoView({ behavior: 'smooth' });
  };
  useEffect(() => {
    // Early bail if we lack either ID
    if (!recipient?._id || !currentUserId) {
      console.warn('ChatBox: missing recipient or current user ID');
      return;
    }

    // 1) Load existing messages
    const fetchMessages = async () => {
      try {
        const res = await api.get('/messages');
        const twoWay = res.data.filter(msg =>
          (msg.sender === currentUserId && msg.recipient ===
recipient._id) ||
          (msg.sender === recipient._id && msg.recipient ===
currentUserId)
        );
        setMessages(twoWay);
        scrollToBottom();
      } catch (err) {
        console.error('Error loading messages:', err);
      }
    };

    fetchMessages();

    // 2) Setup socket listeners
    setupSocket();
    const handler = (msg) => {
      // only append if this message is between us
      if (
        (msg.sender === currentUserId && msg.recipient ===
recipient._id) ||
        (msg.sender === recipient._id && msg.recipient ===
currentUserId)
      ) {
        setMessages(prev => [...prev, msg]);
        scrollToBottom();
      }
    };
    subscribeToMessages(handler);
    subscribeToMessageSent(handler);
    // 3) Cleanup on unmount or recipient change
    return () => {
      disconnectSocket();
      setMessages([]);  // reset chat window
    };

  }, [recipient, currentUserId]);

  const handleSend = (e) => {
    e.preventDefault();
    // Guard again before sending
    if (!input.trim() || !recipient?._id || !currentUserId) {
      console.warn('ChatBox: missing recipient or current user ID');
      return;
    }
    // send over WebSocket (server will tag sender via token)
    sendMessageWS(recipient._id, input.trim());
    setInput('');
  };

  return (
    <div className="chatbox" style={{ border: '1px solid #ccc',
padding: 12, borderRadius: 8 }}>
      <div style={{ height: 300, overflowY: 'auto', marginBottom: 8 }}>
        {messages.map((msg) => (
          <div
            key={msg._id}
            style={{
              textAlign: msg.sender === currentUserId ? 'right' :
'left',
              marginBottom: 4,
            }}
          >
            <span
              style={{
                background: msg.sender === currentUserId ? '#b3d4fc' :
'#eee',
                padding: '6px 10px',
                borderRadius: 12,
                display: 'inline-block',
                maxWidth: '80%',
                wordWrap: 'break-word',
              }}
            >
              {msg.body}
            </span>
          </div>
        ))}
        <div ref={bottomRef} />
      </div>

      <form onSubmit={handleSend} style={{ display: 'flex' }}>
        <input
          type="text"
          value={input}
          onChange={(e) => setInput(e.target.value)}
          placeholder="Type a message..."
          style={{
            flex: 1,
            padding: '8px',
            border: '1px solid #ccc',
            borderRadius: '4px',
            marginRight: '8px',
            fontSize: '14px',
          }}
        />
        <button
          type="submit"
          style={{
            padding: '8px 12px',
            border: '1px solid #ccc',
            borderRadius: '4px',
            cursor: 'pointer',
          }}
        >
          Send
        </button>
      </form>
    </div>
  );
};

export default ChatBox;
```

Payment -Users can select or deselect courses by clicking on them; selected courses are highlighted using conditional styling.The total cost updates automatically as selections change.
When at least one course is selected, users can click a “Download Receipt” button to generate and download a text file containing a formatted payment receipt.

```
// client/src/components/PaymentPage.js
import React, { useState, useEffect } from 'react';
import axios from 'axios';
import './payments.css';

const PaymentPage = () => {
    const [products] = useState([
        { id: 1, name: 'CSE110', price: 22000, description:
'Introduction to Programming' },
        { id: 2, name: 'CSE111', price: 22000, description:
'Object-Oriented Programming' },
        { id: 3, name: 'CSE112', price: 22000, description: 'Data
Structures' }
    ]);
    const [selectedItems, setSelectedItems] = useState([]);
    const [total, setTotal] = useState(0);

    useEffect(() => {
        const calculatedTotal = selectedItems.reduce((sum, item) => sum
+ item.price, 0);
        setTotal(calculatedTotal);
    }, [selectedItems]);

    const toggleSelection = (productId) => {
        setSelectedItems(prevItems => {
            const existingItem = prevItems.find(item => item.id ===
productId);
            if (existingItem) {
                return prevItems.filter(item => item.id !== productId);
            } else {
                const product = products.find(p => p.id === productId);
                return [...prevItems, product];
            }
        });
    };

    const generateReceipt = () => {
        const currentDate = new Date();
        const formattedDate = currentDate.toLocaleDateString('en-US', {
            year: 'numeric',
            month: 'short',
            day: 'numeric'
        });

        const receiptContent = `
    ================================
           COURSE PAYMENT RECEIPT
    ================================
    Date: ${formattedDate}
    Payment Bank: BRAC BANK

    COURSE DETAILS:
    ${selectedItems.map(item =>
        `- ${item.name}: ${item.description}
(৳${item.price.toLocaleString('en-IN')})`
    ).join('\n')}

    TOTAL: ৳${total.toLocaleString('en-IN')}

    Please pay at any BRAC BANK branch
    or through bKash to BRAC BANK account

    Thank you for your enrollment!
    ================================
    `;

        const blob = new Blob([receiptContent], { type: 'text/plain'
});
        const url = URL.createObjectURL(blob);
        const link = document.createElement('a');
        link.href = url;
        link.download = `Payment_Receipt_${currentDate.getTime()}.txt`;
        document.body.appendChild(link);
        link.click();
        document.body.removeChild(link);
        URL.revokeObjectURL(url);
    };
    return (
        <div className="payment-container">
            <div className="payment-card">
                <header className="payment-header">
                    <h1 className="payment-title">Course
Enrollment</h1>
```

```
                    <p className="payment-subtitle">Select your courses
and download the payment receipt</p>
                </header>

                <div className="courses-grid">
                    {products.map(product => (
                        <div
                            key={product.id}
                            className={`course-card
${selectedItems.some(item => item.id === product.id) ? 'selected' :
''}`}
                            onClick={() => toggleSelection(product.id)}
                        >
                            <div className="course-selector">
                                <div className="custom-checkbox">
                                    {selectedItems.some(item => item.id
=== product.id) && (
                                        <div
className="checkmark"></div>
                                    )}
                                </div>
                            </div>
                            <div className="course-content">
                                <h3
className="course-name">{product.name}</h3>
                                <p
className="course-description">{product.description}</p>
                            </div>
                            <div className="course-price">

<span>৳{product.price.toLocaleString('en-IN')}</span>
                            </div>
                        </div>
                    ))}
                </div>

                <div className="payment-footer">
                    <div className="total-display">
                        <span className="total-label">Total
Amount:</span>
                        <span
className="total-amount">৳{total.toLocaleString('en-IN')}</span>
                    </div>
                    <button
                        onClick={generateReceipt}
                        disabled={total <= 0}
                        className="receipt-button"
                    >
                        <span className="button-icon">↓</span>
                        Download Receipt
                    </button>
                </div>
            </div>
        </div>
    );
};

export default PaymentPage;
/* client/src/components/PaymentPage.css */
:root {
    --primary-color: #4361ee;
    --primary-light: #e6f0ff;
    --secondary-color: #3a0ca3;
    --text-dark: #2b2d42;
    --text-light: #8d99ae;
    --background-light: #f8f9fa;
    --white: #ffffff;
    --success-color: #4cc9f0;
    --border-radius: 12px;
    --box-shadow: 0 10px 30px rgba(0, 0, 0, 0.08);
    --transition: all 0.3s ease;
}

* {
    box-sizing: border-box;
    margin: 0;
    padding: 0;
}

.payment-container {
    display: flex;
    justify-content: center;
    align-items: center;
    min-height: 100vh;
    padding: 2rem;
    background: linear-gradient(135deg, #f5f7fa 0%, #dfe7f5 100%);
    font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
}

.payment-card {
    width: 100%;
    max-width: 900px;
    background: var(--white);
    border-radius: var(--border-radius);
    box-shadow: var(--box-shadow);
    overflow: hidden;
    transition: var(--transition);
}

.payment-header {
    padding: 2rem;
    background: linear-gradient(135deg, var(--primary-color) 0%,
var(--secondary-color) 100%);
    color: var(--white);
    text-align: center;
}

.payment-title {
    font-size: 2rem;
    font-weight: 700;
    margin-bottom: 0.5rem;
    letter-spacing: 0.5px;
}

.payment-subtitle {
    font-size: 1rem;
    opacity: 0.9;
    font-weight: 400;
}

.courses-grid {
    display: grid;
    grid-template-columns: repeat(auto-fill, minmax(280px, 1fr));
    gap: 1.5rem;
    padding: 2rem;
}

.course-card {
    display: flex;
    flex-direction: column;
    background: var(--white);
    border-radius: var(--border-radius);
    border: 1px solid #e9ecef;
    overflow: hidden;
    cursor: pointer;
    transition: var(--transition);
    position: relative;
}

.course-card:hover {
    transform: translateY(-5px);
    box-shadow: 0 15px 30px rgba(0, 0, 0, 0.1);
}

.course-card.selected {
    border: 2px solid var(--primary-color);
    background: var(--primary-light);
}

.course-selector {
    padding: 1rem;
    display: flex;
    align-items: center;
}

.custom-checkbox {
    width: 22px;
    height: 22px;
    border: 2px solid var(--text-light);
    border-radius: 6px;
    display: flex;
    align-items: center;
    justify-content: center;
    transition: var(--transition);
}

.course-card.selected .custom-checkbox {
    border-color: var(--primary-color);
    background: var(--primary-color);
}

.checkmark {
    width: 12px;
    height: 12px;
    background: var(--white);
    clip-path: polygon(28% 38%, 41% 53%, 75% 24%, 86% 38%, 40% 78%, 15%
50%);
}

.course-content {
    padding: 0 1rem 1rem;
    flex-grow: 1;
}

.course-name {
    color: var(--text-dark);
    font-size: 1.25rem;
    font-weight: 600;
    margin-bottom: 0.5rem;
}

.course-description {
    color: var(--text-light);
    font-size: 0.9rem;
    line-height: 1.5;
}

.course-price {
    padding: 1rem;
    text-align: right;
    font-size: 1.3rem;
    font-weight: 700;
    color: var(--primary-color);
    border-top: 1px dashed #e9ecef;
}



.payment-footer {
    padding: 1.5rem 2rem;
    background: var(--background-light);
    display: flex;
    justify-content: space-between;
    align-items: center;
    flex-wrap: wrap;
```

```css
    gap: 1rem;
}

.total-display {
    display: flex;
    align-items: center;
    gap: 1rem;
}

.total-label {
    font-size: 1.1rem;
    color: var(--text-light);
}

.total-amount {
    font-size: 1.5rem;
    font-weight: 700;
    color: var(--text-dark);
}

.receipt-button {
    display: inline-flex;
    align-items: center;
    gap: 0.5rem;
    padding: 0.8rem 1.8rem;
    background: var(--primary-color);
    color: var(--white);
    border: none;
    border-radius: 50px;
    font-size: 1rem;
    font-weight: 600;
    cursor: pointer;
    transition: var(--transition);
    position: relative;
    overflow: hidden;
}

.receipt-button:hover {
    background: var(--secondary-color);
    transform: translateY(-2px);
    box-shadow: 0 5px 15px rgba(67, 97, 238, 0.3);
}
```

```css
.receipt-button:disabled {
    background: #adb5bd;
    cursor: not-allowed;
    transform: none;
    box-shadow: none;
}

.receipt-button::after {
    content: '';
    position: absolute;
    top: 50%;
    left: 50%;
    width: 5px;
    height: 5px;
    background: rgba(255, 255, 255, 0.5);
    opacity: 0;
    border-radius: 100%;
    transform: scale(1, 1) translate(-50%);
    transform-origin: 50% 50%;
}

.receipt-button:focus:not(:active)::after {
    animation: ripple 1s ease-out;
}

@keyframes ripple {
    0% {
        transform: scale(0, 0);
        opacity: 0.5;
    }

    100% {
        transform: scale(20, 20);
        opacity: 0;
    }
}

.button-icon {
    font-size: 1.2rem;
    font-weight: bold;
}

@media (max-width: 768px) {
    .payment-container {
        padding: 1rem;
    }

    .payment-header {
        padding: 1.5rem;
    }

    .courses-grid {
        grid-template-columns: 1fr;
        padding: 1.5rem;
    }

    .payment-footer {
        flex-direction: column;
        align-items: stretch;
    }
}
```

Report-The Reports component provides a tabbed dashboard for viewing and exporting system reports related to users, courses, and bookings. It dynamically fetches and displays data in tables based on the selected tab using the Bootstrap Tabs component. Each report can be exported as a CSV file using a blob download. The component also handles loading states and empty datasets with proper UI feedback.

```
import React, { useState, useEffect } from 'react';
import api from '../services/api';
import { Tab, Tabs, Table, Button } from 'react-bootstrap';

const Reports = () => {
  const [activeTab, setActiveTab] = useState('users');
  const [reportData, setReportData] = useState({ users: [], courses:
[], bookings: [] });
  const [loading, setLoading] = useState(false);

  const fetchReportData = async (type) => {
    setLoading(true);
    try {
      const res = await api.get(`/admin/reports/${type}`);
      setReportData(prev => ({ ...prev, [type]: res.data.data || []
}));
    } catch (err) {
      console.error('Failed to fetch report data:', err);
    } finally {
      setLoading(false);
    }
  };

  const downloadCSV = async (type) => {
    try {
      const res = await api.get(`/admin/reports/${type}?format=csv`, {
        responseType: 'blob' // Important to get file blob
      });

      const blob = new Blob([res.data], { type: 'text/csv' });
      const url = window.URL.createObjectURL(blob);
      const link = document.createElement('a');
      link.href = url;
      link.setAttribute('download', `${type}_report_${new
Date().toISOString().split('T')[0]}.csv`);
      document.body.appendChild(link);
      link.click();
      link.remove();
    } catch (err) {
      console.error(`CSV export failed for ${type}:`, err);
    }
  };

  useEffect(() => {
    fetchReportData(activeTab);
  }, [activeTab]);

  const renderTable = () => {
    const data = reportData[activeTab];

    // Check if data is valid (non-null and non-empty)
    if (!data || data.length === 0) {
      return <p>No data available</p>;
    }

    const columns = Object.keys(data[0]);
    return (
      <Table striped bordered hover responsive>
        <thead>
          <tr>
            {columns.map((key, idx) => (
              <th key={idx}>{key.toUpperCase()}</th>
            ))}
          </tr>
        </thead>
        <tbody>
          {data.map((row, i) => (
            <tr key={i}>
              {columns.map((key, j) => (
                <td key={j}>
                  {typeof row[key] === 'object' && row[key] !== null
                    ? JSON.stringify(row[key])
                    : row[key]}
                </td>
              ))}
            </tr>
          ))}
        </tbody>
      </Table>
    );
  };


  return (
    <div className="mt-4 px-4">
      <h2 className="mb-3">System Reports</h2>
      <Tabs activeKey={activeTab} onSelect={(k) => setActiveTab(k)}
className="mb-3">
        <Tab eventKey="users" title="User Reports">
          <Button variant="success" className="mb-3" onClick={() =>
downloadCSV('users')}>
            Export to CSV
          </Button>
          {renderTable()}
        </Tab>
        <Tab eventKey="courses" title="Course Reports">
          <Button variant="success" className="mb-3" onClick={() =>
downloadCSV('courses')}>
            Export to CSV
          </Button>
          {renderTable()}
        </Tab>
        <Tab eventKey="bookings" title="Booking Reports">
          <Button variant="success" className="mb-3" onClick={() =>
downloadCSV('bookings')}>
            Export to CSV
          </Button>
          {renderTable()}
        </Tab>
      </Tabs>
      {loading && <div className="text-center">Loading report
data...</div>}
    </div>
  );
};

export default Reports;
```

Manage Users-This component handles the admin interface for managing users, supporting adding, editing, viewing, and deleting users. It fetches user data from the backend and displays it in a list, with conditional rendering based on the mode (add, edit, or default). Forms are dynamically adapted to the mode—showing or hiding fields like password when editing. Navigation is handled using React Router's useNavigate and useParams for seamless transitions between views.

```
import React, { useEffect, useState } from 'react';
import { useNavigate, useParams } from 'react-router-dom';
import api from '../services/api';
import './styling.css'; // Unified styling

const ManageUsers = ({ mode }) => {
  const [users, setUsers] = useState([]);
  const [newUser, setNewUser] = useState({ name: '', email: '',
password: '', role: '' });
  const [editUser, setEditUser] = useState({ name: '', email: '', role:
'' });
  const { userId } = useParams();
  const navigate = useNavigate();

  const fetchUsers = async () => {
    try {
      const res = await api.get('/admin/users');
      setUsers(res.data);
    } catch (err) {
      console.error('Failed to fetch users:', err);
    }
  };

  useEffect(() => {
    fetchUsers();
  }, []);

  useEffect(() => {
    if (mode === 'edit' && userId) {
      const u = users.find((u) => u._id === userId);
      if (u) setEditUser({ name: u.name, email: u.email, role: u.role
});
    }
  }, [mode, userId, users]);

  const handleAdd = async (e) => {
    e.preventDefault();
    try {
      await api.post('/admin/users', newUser);
      setNewUser({ name: '', email: '', password: '', role: '' });
      fetchUsers();
      navigate('/admin/users');
    } catch (err) {
      console.error('Failed to add user:', err);
    }
  };

  const handleUpdate = async (e) => {
    e.preventDefault();
    try {
      await api.put(`/admin/users/${userId}`, editUser);
      setEditUser({ name: '', email: '', role: '' });
      fetchUsers();
      navigate('/admin/users');
    } catch (err) {
      console.error('Failed to update user:', err);
    }
  };

  const handleDelete = async (id) => {
    try {
      await api.delete(`/admin/users/${id}`);
      fetchUsers();
    } catch (err) {
      console.error('Failed to delete user:', err);
    }
  };

  const renderForm = (isEdit = false) => {
    const user = isEdit ? editUser : newUser;
    const setUser = isEdit ? setEditUser : setNewUser;
    const handleSubmit = isEdit ? handleUpdate : handleAdd;

    return (
      <div className="manage-users-container">
        <h2>{isEdit ? 'Edit User' : 'Add User'}</h2>
        <form onSubmit={handleSubmit} className="user-form">
          <input
            value={user.name}
            onChange={(e) => setUser({ ...user, name: e.target.value
})}
            placeholder="Name"
            required
          />
          <input
            value={user.email}
            onChange={(e) => setUser({ ...user, email: e.target.value
})}
            placeholder="Email"
            required
          />
          {!isEdit && (
            <input
              value={user.password}
              onChange={(e) => setUser({ ...user, password:
e.target.value })}
              placeholder="Password"
              required
            />
          )}
          <input
            value={user.role}
            onChange={(e) => setUser({ ...user, role: e.target.value
})}
            placeholder="Role"
            required
          />
          <div className="form-buttons">
            <button type="submit" className={`btn ${isEdit ?
'btn-update' : 'btn-add'}`}>
              {isEdit ? 'Update User' : 'Add User'}
            </button>
            <button
              type="button"
              className="btn btn-cancel"
              onClick={() => navigate('/admin/users')}
            >
              Cancel
            </button>
          </div>
        </form>
      </div>
    );
  };

  if (mode === 'add') return renderForm(false);
  if (mode === 'edit') return renderForm(true);

  return (
    <div className="manage-users-container">
      <h2>Manage Users</h2>
      <button className="btn btn-add" onClick={() =>
navigate('/admin/users/add')}>
        Add New User
      </button>
      <ul className="user-list">
        {users.map((user) => (
          <li key={user._id} className="user-item">
            <strong>{user.name}</strong> ({user.email}) - {user.role}
            <div className="user-actions">
              <button
                className="btn btn-edit"
                onClick={() =>
navigate(`/admin/users/edit/${user._id}`)}
              >
                Edit
              </button>
              <button
```

```
                className="btn btn-delete"
                onClick={() => handleDelete(user._id)}
              >
                Delete
              </button>
            </div>
          </li>
        ))}
      </ul>
    </div>
  );
};

export default ManageUsers;
```

6. Github Repo [Public] Link

https://tinyurl.com/4fmatznx

7. Link of Deployed Project

8. Individual Contribution

Group member - 01	
Name: Yasin Rahman	Student ID: 23341115
Functional Requirements which are developed by this member:	
1. Admins can create, update and delete courses.	
2. Admins can send notifications to users about any change.	
3. Payment portal from which users can connect to the university payment portal and pay in installments or in full.	
4. The system will have the functionality of connecting and allowing users to interact with the university library, transportation and club activities, etc.	
5. Show all slots available and users can filter using their preferred time	

6. System will show scheduled matches so users can book a versus match.

Group member - 02	
Name: Stanley Matthew Das	Student ID: 21141014
Functional Requirements which are developed by this member:	
1. Admins can view and manage field bookings such as resolving conflicts and cancelling bookings.	
2. Admins can generate reports of data.	
3. The system will allow profile customisation, such as, uploading profile pictures, etc.	
4. Users can view the course materials and interact with popquiz, etc	
5. System will display the progress as a percentage bar.	

Group member - 03	
Name: Arif Jawad Alvi	Student ID: 21301039
Functional Requirements which are developed by this member:	
1. Users can request something to the University Board via a form.	
2. The system will allow users to communicate with each other through in-app messaging.	
3. System will display a list of all available courses with details, where users can search and filter courses.	
4. Users can enroll into a course but the system will prevent enrollment if the user is currently enrolled into it.	
5. The system will allow cancellations before 24 hours from the booked match time. A cancellation notification will be sent to the users who booked using in-app messaging.	

Group member - 04	
Name:	Student ID:
Functional Requirements which are developed by this member:	
1.	
2.	

3.
4.

9. References