addinternship.js

```
import React, { useState } from "react";
import axios from "axios";
import { useNavigate } from "react-router-dom";
import "./AddInternship.css";

const AddInternship = () => {
  const [formData, setFormData] = useState({
    title: "",
    company: "",
    location: "",
    description: "",
    stipend: "",
    duration: "",
    applicationLink: "",
  });

  const [error, setError] = useState("");
  const [loading, setLoading] = useState(false);
  const navigate = useNavigate();

  // Handle Input Change
  const handleChange = (e) => {
    setFormData({ ...formData, [e.target.name]: e.target.value });
  };

  // Handle Form Submission
  const handleSubmit = async (e) => {
    e.preventDefault();
    setError("");
    setLoading(true);

    const token = localStorage.getItem("token");

    if (!token) {
      setError("No token found. Please log in again.");
      navigate("/login");
      return;
    }

    try {
      const response = await axios.post(
        "http://localhost:5000/api/internships/add",
        formData,
        {
          headers: {
            Authorization: `Bearer ${token}`,
```

```
            "Content-Type": "application/json", // Include token in headers
          },
        }
      );

      alert("Internship added successfully!");
      setFormData({
        title: "",
        company: "",
        location: "",
        description: "",
        stipend: "",
        duration: "",
        applicationLink: "",
      });

      // navigate("/internships"); // Redirect to internships page
    } catch (error) {
      console.error("Error adding internship:", error.response?.data?.error ||
error.message);
      setError(error.response?.data?.error || "Failed to add internship. Please try
again.");
    } finally {
      setLoading(false);
    }
  };

  return (
    <div className="add-internship-container">
      <h2>Add Internship</h2>
      {error && <p className="error-message">{error}</p>}

      <form onSubmit={handleSubmit}>
        <input
          type="text"
          name="title"
          placeholder="Title"
          value={formData.title}
          onChange={handleChange}
          required
        />
        <input
          type="text"
          name="company"
          placeholder="Company"
          value={formData.company}
          onChange={handleChange}
          required
        />
        <input
```

```
          type="text"
          name="location"
          placeholder="Location"
          value={formData.location}
          onChange={handleChange}
          required
        />
        <textarea
          name="description"
          placeholder="Description"
          value={formData.description}
          onChange={handleChange}
          required
        />
        <input
          type="text"
          name="stipend"
          placeholder="Stipend"
          value={formData.stipend}
          onChange={handleChange}
          required
        />
        <input
          type="text"
          name="duration"
          placeholder="Duration"
          value={formData.duration}
          onChange={handleChange}
          required
        />

        <input
          type="url"
          name="applicationLink"
          placeholder="Application Link"
          value={formData.applicationLink}
          onChange={handleChange}
          required
        />
        <button type="submit" disabled={loading}>
          {loading ? "Adding..." : "Add Internship"}
        </button>
      </form>
    </div>
  );
};

export default AddInternship;
```

viewinternship.js

```
import React, { useState, useEffect } from "react";
import axios from "axios";
import { FaBookmark, FaRegBookmark } from "react-icons/fa";
import "./ViewInternships.css";
import { useNavigate } from "react-router-dom";

const ViewInternships = () => {
  const navigate = useNavigate();
  const [internships, setInternships] = useState([]);
  const [loading, setLoading] = useState(true);
  const [error, setError] = useState("");
  const [searchQuery, setSearchQuery] = useState("");
  const [filters, setFilters] = useState({ stipend: "", duration: "" });
  const [bookmarks, setBookmarks] = useState([]);
  const [myReferrals, setMyReferrals] = useState([]);

  useEffect(() => {
    const fetchData = async () => {
      try {
        await fetchInternships();
        await fetchBookmarks();
        await fetchMyReferrals();
      } catch (err) {
        console.error("Error loading data:", err);
        setError("Failed to load data. Please try again.");
      } finally {
        setLoading(false);
      }
    };
    fetchData();
  }, []);

  const fetchInternships = async () => {
    try {
      const response = await
axios.get("http://localhost:5000/api/internships/all");
      setInternships(response.data);
    } catch (err) {
      console.error("Error fetching internships:", err);
      throw err;
    }
  };

  const fetchBookmarks = async () => {
    try {
      const token = localStorage.getItem("token");
      if (!token) return;
```

```
    const response = await axios.get(
      "http://localhost:5000/api/bookmarks",
      { headers: { Authorization: `Bearer ${token}` } }
    );
    setBookmarks(response.data);
  } catch (err) {
    console.error("Error fetching bookmarks:", err);
    // Continue even if bookmarks fail
  }
};

const fetchMyReferrals = async () => {
  try {
    const token = localStorage.getItem("token");
    const userId = localStorage.getItem("userId");
    if (!token || !userId) return;

    const response = await axios.get(
      `http://localhost:5000/api/referrals/student/${userId}`,
      { headers: { Authorization: `Bearer ${token}` } }
    );
    setMyReferrals(response.data);
  } catch (err) {
    console.error("Error fetching referrals:", err);
    // Continue even if referrals fail
  }
};

const isBookmarked = (id) => {
  return bookmarks.some((bookmark) => bookmark.internship?._id === id);
};

const getReferralStatus = (internshipId) => {
  const referral = myReferrals.find(r => r.internship?._id === internshipId);
  return referral ? referral.status : null;
};

const handleBookmark = async (id) => {
  try {
    const token = localStorage.getItem("token");
    if (!token) {
      alert("Please log in to bookmark internships.");
      return;
    }

    if (isBookmarked(id)) {
      await axios.delete(
        `http://localhost:5000/api/bookmarks/${id}`,
        { headers: { Authorization: `Bearer ${token}` } }
```

```
        );
        setBookmarks(bookmarks.filter((bookmark) => bookmark.internship?._id !==
id));
      } else {
        const response = await axios.post(
          "http://localhost:5000/api/bookmarks",
          { internshipId: id },
          { headers: { Authorization: `Bearer ${token}` } }
        );
        setBookmarks([...bookmarks, response.data]);
      }
    } catch (err) {
      console.error("Error bookmarking internship:", err);
    }
  };

  const filteredInternships = internships.filter((internship) => {
    const matchesSearchQuery =
      internship.title?.toLowerCase().includes(searchQuery.toLowerCase()) ||
      internship.company?.toLowerCase().includes(searchQuery.toLowerCase());

    const matchesFilters =
      (!filters.stipend || internship.stipend >= filters.stipend) &&
      (!filters.duration || internship.duration === filters.duration);

    return matchesSearchQuery && matchesFilters;
  });

  const handleFilterChange = (e) => {
    const { name, value } = e.target;
    setFilters({ ...filters, [name]: value });
  };

  if (loading) {
    return <div className="loading-state">Loading internships...</div>;
  }

  if (error) {
    return <div className="error-message">{error}</div>;
  }

  return (
    <div className="internships-container">
      <h2>Available Internships</h2>
      <div className="search-bar-container">

      <input
        type="text"
        placeholder="Search internships..."
        value={searchQuery}
```

```
        onChange={(e) => setSearchQuery(e.target.value)}
        className="search-bar"
      />
      </div>
      {filteredInternships.length === 0 ? (
        <p>No internships found.</p>
      ) : (
        <ul className="internships-list">
          {filteredInternships.map((internship) => {
            const referralStatus = getReferralStatus(internship._id);

            return (
              <li key={internship._id} className="internship-item">
                <h3>{internship.title}</h3>
                <p><strong>Company:</strong> {internship.company}</p>
                <p><strong>Location:</strong> {internship.location}</p>
                <p><strong>Description:</strong> {internship.description}</p>
                <p><strong>Stipend:</strong> {internship.stipend}</p>
                <p><strong>Duration:</strong> {internship.duration}</p>
                <p><strong>Application Link:</strong> <a
href={internship.applicationLink} target="_blank" rel="noopener
noreferrer">Apply Here</a></p>
                <p><strong>Posted By:</strong> {internship.postedBy?.name}</p>

                <div className="internship-actions">
                  {/* <button
                    className="bookmark-button"
                    onClick={() => handleBookmark(internship._id)}
                  >
                    {isBookmarked(internship._id) ? <FaBookmark /> : <FaRegBookmark
/>}
                  </button> */}

                  {referralStatus ? (
                    <div className="referral-status-container">
                      <span className={`referral-status
${referralStatus.toLowerCase()}`}>
                        Referral {referralStatus}
                      </span>
                      {referralStatus === "Pending" && (
                        <button
                          onClick={() => navigate('/student/referrals')}
                          className="view-request-btn"
                        >
                          View Request
                        </button>
                      )}
                    </div>
                  ) : (
                    <button
```

```
                      onClick={() =>
navigate(`/request-referral/${internship._id}`)}
                      className="request-referral-btn"
                    >
                      Request Referral
                    </button>
                  )}
                </div>
              </li>
            );
          })}
        </ul>
      )}
    </div>
  );
};

export default ViewInternships;
```

internshipcontroller

```
const Internship = require("../models/Internship");

// ☑ Alumni - Add Internship
const addInternship = async (req, res) => {
  try {
    console.log("Request Body:", req.body);
    console.log("User ID from token:", req.user?.id);

    const { title, company, location, description, stipend, duration, category } =
req.body;

    if (!title || !company || !location || !description || !duration || !category)
{
      return res.status(400).json({ error: "All fields are required except
stipend." });
    }

    if (!req.user) {
      return res.status(401).json({ error: "Unauthorized. User not found." });
    }

    const newInternship = new Internship({
      title,
```

```
      company,
      location,
      description,
      stipend,
      duration,
      category,
      postedBy: req.user.id,
    });

    console.log("New Internship Object:", newInternship);

    await newInternship.save();
    console.log("Internship saved successfully.");
    res.status(201).json({ message: "Internship added successfully" });
  } catch (err) {
    console.error("Error adding internship:", err.message);
    res.status(500).json({ error: err.message });
  }
};

// ☑ Students - Get All Internships
const getAllInternships = async (req, res) => {
  try {
    const internships = await Internship.find().populate("postedBy", "name
company");
    res.status(200).json(internships);
  } catch (err) {
    res.status(500).json({ error: err.message });
  }
};

// ☑ Get Internship Details by ID
const getInternshipById = async (req, res) => {
  try {
    const internship = await
Internship.findById(req.params.id).populate("postedBy", "name");
    if (!internship) {
      return res.status(404).json({ message: "Internship not found" });
    }
    res.json(internship);
  } catch (err) {
    res.status(500).json({ message: "Failed to fetch internship details" });
  }
};

module.exports = { addInternship, getAllInternships, getInternshipById };
```

models/internship

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



const internshipSchema = new mongoose.Schema({
  title: { type: String, required: true },
  company: { type: String, required: true },
  location: { type: String, required: true },
  description: { type: String, required: true },
  stipend: { type: String, required: true },
  duration: { type: String, required: true },
  applicationLink: { type: String, required: true },

  postedBy: { type: mongoose.Schema.Types.ObjectId, ref: 'User', required: true },
// Reference to the user who posted the internship
  createdAt: { type: Date, default: Date.now },
});

module.exports = mongoose.model('Internship', internshipSchema);
```

internshipRoutes

```
const express = require("express");
const router = express.Router();
const mongoose = require("mongoose");
const Internship = require("../models/Internship");
const authMiddleware = require("../middleware/authMiddleware");

// Alumni - Add Internship (unchanged)
router.post("/add", authMiddleware, async (req, res) => {
    try {
        const { title, company, location, description, stipend, duration,
applicationLink } = req.body;
        const userId = req.user.userId;
```

```
        if (!userId) {
            return res.status(401).json({ error: "Unauthorized: No user ID found."
});
        }

        const newInternship = new Internship({
            title,
            company,
            location,
            description,
            stipend,
            duration,
            applicationLink,
            postedBy: userId,
        });

        await newInternship.save();
        res.status(201).json({ message: "Internship added successfully!",
internship: newInternship });
    } catch (error) {
        res.status(500).json({ error: error.message });
    }
});

// Students - Get All Internships (unchanged)
router.get("/all", async (req, res) => {
    try {
        const internships = await Internship.find().populate("postedBy", "name
company");
        res.status(200).json(internships);
    } catch (err) {
        res.status(500).json({ error: err.message });
    }
});

// Get internship details by ID (optimized version)
router.get("/:id", async (req, res) => {
    try {
        // Validate the ID format first
        if (!mongoose.Types.ObjectId.isValid(req.params.id)) {
            return res.status(400).json({ message: "Invalid internship ID format"
});
        }

        const internship = await Internship.findById(req.params.id)
            .populate("postedBy", "_id name email role")
            .lean();

        if (!internship) {
            console.log("Internship not found:", req.params.id);
```

```
            return res.status(404).json({ message: "Internship not found" });
        }

        // Verify alumni data exists
        if (!internship.postedBy || !internship.postedBy._id) {
            console.error("Missing alumni data for internship:", internship._id);
            return res.status(400).json({
                message: "No valid alumni associated with this internship",
                debug: process.env.NODE_ENV === 'development' ? { internship } :
undefined
            });
        }

        // Prepare response data - keeping all existing fields
        const responseData = {
            ...internship,
            // Ensure alumniId is always included as a string
            alumniId: internship.postedBy._id.toString(),
            // Maintain the alumni object structure
            alumni: {
                id: internship.postedBy._id,
                name: internship.postedBy.name,
                email: internship.postedBy.email,
                role: internship.postedBy.role
            },
            // Keep original postedBy reference if needed by other services
            postedBy: internship.postedBy
        };

        res.json(responseData);

    } catch (error) {
        console.error("Error fetching internship:", {
            error: error.message,
            stack: error.stack,
            params: req.params
        });
        res.status(500).json({
            message: "Failed to fetch internship details",
            error: process.env.NODE_ENV === 'development' ? error.message :
undefined
        });
    }
});

module.exports = router
```

referralrequestform

```
import React, { useState, useEffect } from 'react';
import { useNavigate, useParams } from 'react-router-dom';
import axios from 'axios';
import "./ReferralRequestForm.css";

const ReferralRequestForm = () => {
  const { internshipId } = useParams();
  const [message, setMessage] = useState('');
  const [loading, setLoading] = useState(false);
  const [fetching, setFetching] = useState(true);
  const [success, setSuccess] = useState(false);
  const [alumniId, setAlumniId] = useState(null);
  const [resume, setResume] = useState(null);
  const [error, setError] = useState('');
  const navigate = useNavigate();

  useEffect(() => {
    const fetchInternshipDetails = async () => {
      try {
        const response = await
axios.get(`http://localhost:5000/api/internships/${internshipId}`);
        const fetchedAlumniId = response.data.alumniId ||
                                response.data.postedBy?._id ||
                                response.data.postedBy;

        if (!fetchedAlumniId) {
          throw new Error('No alumni associated with this internship');
        }

        setAlumniId(fetchedAlumniId);
        setFetching(false);
      } catch (err) {
        console.error('Error fetching internship:', err);
        setError(err.response?.data?.error || err.message || 'Failed to load
internship details');
        setFetching(false);
        navigate('/internships');
      }
```

```
    };

    fetchInternshipDetails();
  }, [internshipId, navigate]);

  const handleSubmit = async (e) => {
    e.preventDefault();
    setLoading(true);
    setError('');

    try {
      const token = localStorage.getItem('token');
      const userId = localStorage.getItem('userId');

      if (!token || !userId) {
        throw new Error('Authentication required');
      }

      if (!resume) {
        throw new Error('Resume required');
      }

      if (!alumniId) {
        throw new Error('No valid alumni recipient found');
      }

      const formData = new FormData();
      formData.append('studentId', userId);
      formData.append('alumniId', alumniId);
      formData.append('internshipId', internshipId);
      formData.append('message', message);
      formData.append('resume', resume);

      const response = await axios.post('http://localhost:5000/api/referrals',
formData, {
        headers: {
          'Authorization': `Bearer ${token}`,
          'Content-Type': 'multipart/form-data'
        }
      });

      sessionStorage.setItem("referralSubmitted", "true");
      setSuccess(true);
      setTimeout(() => {
        navigate('/student/referrals', { state: { refresh: true } });
      }, 1500);
    } catch (err) {
      console.error('Submission error:', err);
      if (err.response?.data?.error === "You have already requested a referral for
this internship") {
```

```
        setError(`You've already requested a referral for this internship (Status:
${err.response.data.existingStatus})`);
        setTimeout(() => {
          navigate('/student/referrals');
        }, 3000);
      } else {
        setError(err.response?.data?.error || err.message || 'Request failed');
      }
    } finally {
      setLoading(false);
    }
  };

  const handleFileChange = (e) => {
    const file = e.target.files[0];
    if (!file) return;

    const validTypes = [
      'application/pdf',
      'application/msword',
      'application/vnd.openxmlformats-officedocument.wordprocessingml.document'
    ];

    if (!validTypes.includes(file.type)) {
      setError('Only PDF/DOC/DOCX files allowed');
      return;
    }

    if (file.size > 5 * 1024 * 1024) {
      setError('File must be smaller than 5MB');
      return;
    }

    setResume(file);
    setError('');
  };

  if (fetching) return <div className="loading-state">Loading internship
details...</div>;
  if (error && !error.includes("already requested")) return <div
className="error-message">{error}</div>;

  return (
    <div className="referral-form-container">
      <h2 className="referral-form-title">Request Referral</h2>

      {success ? (
        <div className="success-message">
          <p>Request sent successfully!</p>
        </div>
```

```
      ) : (
        <form onSubmit={handleSubmit} className="referral-form">
          {error && (
            <div className={`error-message ${error.includes("already requested") ?
"warning-message" : ""}`}>
              {error}
              {error.includes("already requested") && (
                <p>Redirecting to your referrals page...</p>
              )}
            </div>
          )}

          <div className="form-group">
            <label className="form-label">Why should you be referred?</label>
            <textarea
              value={message}
              onChange={(e) => setMessage(e.target.value)}
              className="form-textarea"
              rows="5"
              required
              placeholder="Explain why you're a good fit for this opportunity..."
              disabled={error.includes("already requested")}
            />
          </div>

          <div className="form-group">
            <label className="form-label">Upload Resume</label>
            <div className="file-input-container">
              <label className="file-input-label">
                <input
                  type="file"
                  onChange={handleFileChange}
                  className="file-input"
                  accept=".pdf,.doc,.docx"
                  required
                  disabled={error.includes("already requested")}
                />
                <div className="file-input-text">
                  <span className="file-input-icon">📄</span>
                  <span>{resume ? 'Change file' : 'Click to upload'}</span>
                  <span className="file-hint">PDF, DOC, or DOCX (Max 5MB)</span>
                </div>
              </label>
              {resume && (
                <p className="file-selected">Selected: {resume.name}</p>
              )}
            </div>
          </div>

          <button
```

```
            type="submit"
            disabled={loading || !message.trim() || !resume || !alumniId ||
error.includes("already requested")}
            className="submit-btn"
          >
            {loading ? 'Sending...' : 'Send Request'}
          </button>
        </form>
      )}
    </div>
  );
};

export default ReferralRequestForm;
```

alumnireferrallist

```
import React, { useState, useEffect } from "react";
import axios from "axios";
import "./AlumniReferralList.css";

const AlumniReferralList = () => {
  const [referrals, setReferrals] = useState([]);
  const [loading, setLoading] = useState(true);
  const [error, setError] = useState(null);
  const [selectedReferral, setSelectedReferral] = useState(null);
  const [rejectReason, setRejectReason] = useState("");
  const [shouldRefresh, setShouldRefresh] = useState(false);
  const [isUpdating, setIsUpdating] = useState(false);

  const getAlumniReferrals = async (alumniId) => {
    try {
      const response = await axios.get(
        `http://localhost:5000/api/referrals/alumni/${alumniId}`,
        {
          headers: {
            Authorization: `Bearer ${localStorage.getItem("token")}`
          }
        }
      );
      return response.data;
    } catch (error) {
      throw error.response?.data || error;
```

```
    }
  };

  const viewResume = (filename) => {
    window.open(`http://localhost:5000/api/referrals/resume/${filename}`, '_blank',
'noopener,noreferrer');
  };

  useEffect(() => {
    const fetchReferrals = async () => {
      try {
        const alumniId = localStorage.getItem("userId");
        console.log("Alumni ID from localStorage:", alumniId);

        if (!alumniId) throw new Error("User ID not found");

        const referrals = await getAlumniReferrals(alumniId);
        setReferrals(referrals);
        setError(null);
      } catch (error) {
        console.error("Failed to fetch referrals:", error);
        setError(error.message);
      } finally {
        setLoading(false);
        setShouldRefresh(false);
      }
    };

    fetchReferrals();

    // Check for new referrals periodically (every 30 seconds)
    const interval = setInterval(fetchReferrals, 30000);
    return () => clearInterval(interval);
  }, [shouldRefresh]);

  const handleStatusUpdate = async (status) => {
    setIsUpdating(true);
    try {
      // For rejections, we need a reason
      if (status === "Rejected") {
        let reason = rejectReason;

        // If no reason in state, prompt user
        if (!reason.trim()) {
          reason = window.prompt("Please enter the reason for rejection:");
          if (!reason || !reason.trim()) {
            alert("Rejection reason is required");
            setIsUpdating(false);
            return;
          }
```

```
    }

    // Make the API call with the reason
    await axios.put(
      `http://localhost:5000/api/referrals/${selectedReferral._id}/status`,
      {
        status,
        rejectReason: reason
      },
      {
        headers: {
          Authorization: `Bearer ${localStorage.getItem("token")}`
        }
      }
    );

    // Update local state
    setReferrals(referrals.map(ref =>
      ref._id === selectedReferral._id ? {
        ...ref,
        status,
        rejectReason: reason
      } : ref
    ));
  } else {
    // For acceptances
    await axios.put(
      `http://localhost:5000/api/referrals/${selectedReferral._id}/status`,
      { status },
      {
        headers: {
          Authorization: `Bearer ${localStorage.getItem("token")}`
        }
      }
    );

    setReferrals(referrals.map(ref =>
      ref._id === selectedReferral._id ? { ...ref, status } : ref
    ));
  }

  // Reset modal state
  setSelectedReferral(null);
  setRejectReason("");
  setShouldRefresh(true);
} catch (error) {
  console.error("Failed to update status:", error);
  setError(error.message);
} finally {
  setIsUpdating(false);
```

```
    }
  };

  const handleManualRefresh = () => {
    setLoading(true);
    setShouldRefresh(true);
  };

  if (loading) return <div className="loading-state">Loading...</div>;
  if (error) return <div className="error-state">Error: {error}</div>;

  return (
    <div className="referrals-container">
      <div className="referrals-header">
        <h2 className="referrals-title">Referral Requests</h2>
        <button
          onClick={handleManualRefresh}
          className="refresh-btn"
          disabled={loading}
        >
          {loading ? "Refreshing..." : "Refresh List"}
        </button>
      </div>

      {referrals.length === 0 ? (
        <div className="empty-state">
          <p className="empty-text">No referral requests found</p>
        </div>
      ) : (
        <div className="referrals-grid">
          {referrals.map(referral => (
            <div key={referral._id} className="referral-card">
              {/* <h3 className="card-title">
                {referral.internship.title || referral.internship.position} at
{referral.internship.company}
              </h3> */}
              <div className="space-y-2">
                <p className="card-detail">
                  <span className="card-label">From:</span> {referral.student.name}
                </p>
                <p className="card-detail">
                  <span className="card-label">Message:</span> {referral.message}
                </p>
                <p className="card-detail">
                  <span className="card-label">Status:</span>{" "}
                  <span className={
                    referral.status === "Accepted" ? "status-accepted" :
                    referral.status === "Rejected" ? "status-rejected" :
"status-pending"
                  }>
```

```
                    {referral.status}
                  </span>
                </p>
                {referral.status === "Rejected" && referral.rejectReason && (
                  <p className="card-detail">
                    <span className="card-label">Reason:</span>
{referral.rejectReason}
                  </p>
                )}
              </div>

              {referral.status === "Pending" && (
                <div className="card-actions">
                  <button
                    onClick={() => setSelectedReferral(referral)}
                    className="action-btn action-btn-primary"
                  >
                    Take Action
                  </button>
                  <button
                    onClick={() => viewResume(referral.resumeUrl.split('/').pop())}
                    className="action-btn action-btn-secondary"
                  >
                    View Resume
                  </button>
                </div>
              )}
            </div>
          ))}
        </div>
      )}

      {selectedReferral && (
        <div className="modal-overlay">
          <div className="modal-content">
            <h3 className="modal-title">Update Referral Status</h3>
            <p>Request from {selectedReferral.student.name} for
{selectedReferral.internship.position}</p>

            <div className="modal-actions">
              <button
                onClick={() => handleStatusUpdate("Accepted")}
                className="modal-btn modal-btn-accept"
                disabled={isUpdating}
              >
                {isUpdating ? "Processing..." : "Accept"}
              </button>

              <button
                onClick={() => handleStatusUpdate("Rejected")}
```

```
                  className="modal-btn modal-btn-reject"
                  disabled={isUpdating}
                >
                  {isUpdating ? "Processing..." : "Reject"}
                </button>

                <button
                  onClick={() => {
                    setSelectedReferral(null);
                    setRejectReason("");
                  }}
                  className="modal-btn modal-btn-cancel"
                  disabled={isUpdating}
                >
                  Cancel
                </button>
              </div>
            </div>
          </div>
        )}
      </div>
    );
};

export default AlumniReferralList;
```

studentreferrallist

```
import React, { useState, useEffect } from "react";
import axios from "axios";
import "./StudentReferralList.css";

const StudentReferralList = () => {
  const [referrals, setReferrals] = useState([]);
  const [loading, setLoading] = useState(true);
  const [error, setError] = useState(null);
  const [shouldRefresh, setShouldRefresh] = useState(false);

  const getStudentReferrals = async (studentId) => {
    try {
      const response = await axios.get(
        `http://localhost:5000/api/referrals/student/${studentId}`,
        {
```

```
        headers: {
          Authorization: `Bearer ${localStorage.getItem("token")}`,
        },
      }
    );
    return response.data;
  } catch (error) {
    throw error.response?.data || error;
  }
};

useEffect(() => {
  const fetchReferrals = async () => {
    try {
      const studentId = localStorage.getItem("userId");
      console.log("Student ID from localStorage:", studentId);

      if (!studentId) throw new Error("User ID not found");

      const referrals = await getStudentReferrals(studentId);
      setReferrals(referrals);
      setError(null);
    } catch (error) {
      console.error("Failed to fetch referrals:", error);
      setError(error.message);
    } finally {
      setLoading(false);
      setShouldRefresh(false);
    }
  };

  fetchReferrals();

  const comingFromSubmission = sessionStorage.getItem("referralSubmitted");
  if (comingFromSubmission) {
    sessionStorage.removeItem("referralSubmitted");
    setShouldRefresh(true);
  }
}, [shouldRefresh]);

const handleManualRefresh = () => {
  setLoading(true);
  setShouldRefresh(true);
};

if (loading) return <div className="loading-state">Loading...</div>;
if (error) return <div className="error-state">Error: {error}</div>;

return (
  <div className="referrals-container">
```

```
      <div className="referrals-header">
        <h2 className="referrals-title">Your Referral Requests</h2>
        <button
          onClick={handleManualRefresh}
          className="refresh-btn"
        >
          Refresh List
        </button>
      </div>

      {referrals.length === 0 ? (
        <div className="empty-state">
          <p className="empty-text">You haven't made any referral requests yet</p>
          <p className="empty-hint">
            If you just submitted a request, try refreshing the list.
          </p>
        </div>
      ) : (
        <div className="referrals-grid">
          {referrals.map((referral) => (
            <div key={referral._id} className="referral-card">
              {/* <h3 className="card-title">
                {referral.internship.position} at {referral.internship.company}
              </h3> */}
              <div>
                <p className="card-detail">
                  <span className="card-label">To:</span> {referral.alumni.name}
                </p>
                <p className="card-detail">
                  <span className="card-label">Your Message:</span>
{referral.message}
                </p>
                <p className="card-detail">
                  <span className="card-label">Status:</span>{" "}
                  <span className={
                    referral.status === "Accepted" ? "status-accepted" :
                    referral.status === "Rejected" ? "status-rejected" :
"status-pending"
                  }>
                    {referral.status}
                  </span>
                </p>
                {referral.status === "Rejected" && referral.rejectReason && (
                  <div className="rejection-reason">
                    <p className="rejection-reason-title">Reason:</p>
                    <p>{referral.rejectReason}</p>
                  </div>
                )}
              </div>
            </div>
```

```
          ))}
        </div>
      )}
    </div>
  );
};

export default StudentReferralList;
```

models/referral

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

const referralSchema = new mongoose.Schema({
  student: {
    type: mongoose.Schema.Types.ObjectId,
    ref: "User",
    required: true
  },
  alumni: {
    type: mongoose.Schema.Types.ObjectId,
    ref: "User",
    required: true
  },
  internship: {
    type: mongoose.Schema.Types.ObjectId,
    ref: "Internship",
    required: true
  },
  message: {
    type: String,
    required: true
  },
  resumeUrl: {
    type: String,
    required: true,
    default: '/referral-resumes/default-resume.pdf'
  },
  status: {
    type: String,
    enum: ["Pending", "Accepted", "Rejected"],
    default: "Pending"
  },
  rejectReason: {
```

```
    type: String,
    default: ""
  }
}, {
  timestamps: true,
  toJSON: { virtuals: true },
  toObject: { virtuals: true }
});

// Improved population handling for modern Mongoose
referralSchema.post('find', async function(docs) {
  if (!docs || docs.length === 0) return;

  try {
    await Promise.all(docs.map(async doc => {
      try {
        // Modern Mongoose population (no execPopulate needed)
        await doc.populate([
          {
            path: 'student',
            select: 'name email',
            model: 'User'
          },
          {
            path: 'alumni',
            select: 'name email',
            model: 'User'
          },
          {
            path: 'internship',
            select: 'title company position postedBy',
            model: 'Internship',
            populate: {
              path: 'postedBy',
              select: 'name email',
              model: 'User'
            }
          }
        ]);
      } catch (populateErr) {
        console.error('Population error for referral:', {
          referralId: doc._id,
          error: populateErr.message,
          stack: process.env.NODE_ENV === 'development' ? populateErr.stack :
undefined
        });
        // Optionally attach error to document for frontend handling
        doc.populateError = populateErr.message;
      }
    }));
```

```
  } catch (err) {
    console.error('Global population error:', {
      error: err.message,
      stack: process.env.NODE_ENV === 'development' ? err.stack : undefined
    });
  }
});

// Virtuals for easier frontend access
referralSchema.virtual('studentName').get(function() {
  return this.student?.name || 'Unknown Student';
});

referralSchema.virtual('alumniName').get(function() {
  return this.alumni?.name || 'Unknown Alumni';
});

referralSchema.virtual('internshipTitle').get(function() {
  return this.internship?.title || this.internship?.position || 'Unknown Position';
});

referralSchema.virtual('companyName').get(function() {
  return this.internship?.company || 'Unknown Company';
});

// Add indexes for better query performance
referralSchema.index({ student: 1, status: 1 });
referralSchema.index({ alumni: 1, status: 1 });
referralSchema.index({ internship: 1 });
referralSchema.index({ student: 1, internship: 1 }, { unique: false });

module.exports = mongoose.model("Referral", referralSchema);
```

referralroutes

```
const express = require("express");
const router = express.Router();
const multer = require("multer");
const path = require("path");
const fs = require("fs");
const mongoose = require("mongoose");
const Referral = require("../models/Referral");
```

```
const Internship = require("../models/Internship");
const User = require("../models/User");
const { sendReferralNotification, sendReferralStatusUpdate } =
require('../utils/email');

// Configure multer for file uploads
const storage = multer.diskStorage({
  destination: (req, file, cb) => {
    const uploadDir = path.join(__dirname, '../referral_resumes');
    if (!fs.existsSync(uploadDir)) {
      fs.mkdirSync(uploadDir, { recursive: true });
    }
    cb(null, uploadDir);
  },
  filename: (req, file, cb) => {
    const uniqueSuffix = Date.now() + '-' + Math.round(Math.random() * 1E9);
    cb(null, 'resume-' + uniqueSuffix + path.extname(file.originalname));
  }
});

const upload = multer({
  storage: storage,
  limits: { fileSize: 5 * 1024 * 1024 }, // 5MB limit
  fileFilter: (req, file, cb) => {
    const filetypes = /pdf|doc|docx/;
    const mimetype = filetypes.test(file.mimetype);
    const extname = filetypes.test(path.extname(file.originalname).toLowerCase());

    if (mimetype && extname) {
      return cb(null, true);
    }
    cb(new Error('Only PDF, DOC, and DOCX files are allowed'));
  }
});

// Middleware to validate ObjectIDs
const validateObjectId = (req, res, next, id) => {
  if (!mongoose.Types.ObjectId.isValid(id)) {
    return res.status(400).json({ error: "Invalid ID format" });
  }
  next();
};

// Apply validation to all ID parameters
router.param('id', validateObjectId);
router.param('alumniId', validateObjectId);
router.param('studentId', validateObjectId);
router.param('internshipId', validateObjectId);

// Student sends referral request with resume upload
```

```
router.post("/", upload.single('resume'), async (req, res) => {
  try {
    console.log('Received referral data:', req.body);

    const { studentId, alumniId, internshipId, message } = req.body;

    // Validate required fields
    if (!studentId || !alumniId || !internshipId || !message) {
      if (req.file) fs.unlinkSync(req.file.path);
      return res.status(400).json({ error: "All fields are required" });
    }

    if (!req.file) {
      return res.status(400).json({ error: "Resume file is required" });
    }

    // Check if student already has a referral request for this internship
    const existingReferral = await Referral.findOne({
      student: studentId,
      internship: internshipId
    });

    if (existingReferral) {
      fs.unlinkSync(req.file.path);
      return res.status(400).json({
        error: "You have already requested a referral for this internship",
        existingStatus: existingReferral.status
      });
    }

    // Check if internship exists
    const internship = await Internship.findById(internshipId);
    if (!internship) {
      fs.unlinkSync(req.file.path);
      return res.status(404).json({ error: "Internship not found" });
    }

    // Check if alumni exists and is actually an alumni
    const alumni = await User.findById(alumniId);
    if (!alumni || alumni.role !== "Alumni") {
      fs.unlinkSync(req.file.path);
      return res.status(404).json({ error: "Alumni not found or invalid role" });
    }

    // Check if student exists and is actually a student
    const student = await User.findById(studentId);
    if (!student || student.role !== "Student") {
      fs.unlinkSync(req.file.path);
      return res.status(404).json({ error: "Student not found or invalid role" });
    }
```

```
    // Create new referral with resume URL
    const newReferral = new Referral({
      student: studentId,
      alumni: alumniId,
      internship: internshipId,
      message,
      resumeUrl: `/referral-resumes/${req.file.filename}`,
      status: "Pending"
    });

    await newReferral.save();

    // Send email notification to alumni
    await sendReferralNotification(
      alumni.email,
      student.name,
      internship.position || internship.title,
      internship.company,
      message,
      `${req.protocol}://${req.get('host')}${newReferral.resumeUrl}`
    );

    res.status(201).json(newReferral);
  } catch (error) {
    console.error('Error creating referral:', error);
    if (req.file) fs.unlinkSync(req.file.path);
    res.status(500).json({
      error: error.message,
      details: process.env.NODE_ENV === 'development' ? error.stack : undefined
    });
  }
});

// Resume download endpoint
router.get('/resume/:filename', (req, res) => {
  try {
    const filePath = path.join(__dirname, '../referral_resumes',
req.params.filename);
    if (fs.existsSync(filePath)) {
      res.sendFile(filePath);
    } else {
      res.status(404).json({ error: 'Resume not found' });
    }
  } catch (error) {
    res.status(500).json({ error: error.message });
  }
});

// Alumni gets all referral requests
```

```
router.get("/alumni/:alumniId", async (req, res) => {
  try {
    res.set('Cache-Control', 'no-store, must-revalidate');

    const referrals = await Referral.find({ alumni: req.params.alumniId })
      .populate("student", "name email")
      .populate({
        path: "internship",
        select: "title company position postedBy",
        populate: {
          path: "postedBy",
          select: "name email"
        }
      })
      .sort({ createdAt: -1 });

    // Enhanced debug logging
    console.log('Fetched referrals for alumni:', {
      alumniId: req.params.alumniId,
      count: referrals.length,
      referrals: referrals.map(r => ({
        id: r._id,
        student: r.student?.name || 'Unknown',
        alumni: r.alumni?.toString(), // Since we didn't populate alumni here
        internship: {
          title: r.internship?.title || 'No title',
          position: r.internship?.position || 'No position',
          company: r.internship?.company || 'No company',
          postedBy: r.internship?.postedBy?.name || 'Unknown'
        },
        status: r.status,
        createdAt: r.createdAt,
        updatedAt: r.updatedAt
      })),
      populatedFields: {
        student: true,
        internship: true,
        'internship.postedBy': true
      }
    });

    res.json(referrals);
  } catch (error) {
    console.error('Error fetching alumni referrals:', {
      error: error.message,
      stack: process.env.NODE_ENV === 'development' ? error.stack : undefined,
      params: req.params,
      timestamp: new Date().toISOString()
    });
    res.status(500).json({
```

```
      error: 'Failed to fetch referral requests',
      details: process.env.NODE_ENV === 'development' ? error.message : undefined
    });
  }
});

// Student gets their referral requests
router.get("/student/:studentId", async (req, res) => {
  try {
    res.set('Cache-Control', 'no-store, must-revalidate');

    const referrals = await Referral.find({ student: req.params.studentId })
      .populate("alumni", "name email")
      .populate({
        path: "internship",
        select: "title company location position",
        populate: {
          path: "postedBy",
          select: "name email"
        }
      })
      .sort({ createdAt: -1 });

    res.json(referrals);
  } catch (error) {
    console.error('Error fetching student referrals:', error);
    res.status(500).json({
      error: error.message,
      details: process.env.NODE_ENV === 'development' ? error.stack : undefined
    });
  }
});

// Alumni updates referral status
router.put("/:id/status", async (req, res) => {
  try {
    const { status, rejectReason } = req.body;

    if (!["Accepted", "Rejected"].includes(status)) {
      return res.status(400).json({ error: "Invalid status value" });
    }

    if (status === "Rejected" && !rejectReason?.trim()) {
      return res.status(400).json({ error: "Rejection reason is required" });
    }

    const referral = await Referral.findByIdAndUpdate(
      req.params.id,
      {
        status,
```

```
        rejectReason: status === "Rejected" ? rejectReason : undefined,
        updatedAt: Date.now()
      },
      { new: true, runValidators: true }
    )
      .populate("student", "name email")
      .populate("internship", "position company title")
      .populate("alumni", "name email");

    if (!referral) {
      return res.status(404).json({ error: "Referral not found" });
    }

    await sendReferralStatusUpdate(
      referral.student.email,
      referral.alumni.name,
      referral.internship.position || referral.internship.title,
      referral.internship.company,
      status,
      rejectReason
    );

    res.json(referral);
  } catch (error) {
    console.error('Error updating referral status:', error);
    res.status(500).json({
      error: error.message,
      details: process.env.NODE_ENV === 'development' ? error.stack : undefined
    });
  }
});

module.exports = router;
```

email.js

```
const nodemailer = require('nodemailer');
require('dotenv').config();

// Configure nodemailer transporter
const transporter = nodemailer.createTransport({
  service: 'gmail',
  auth: {
    user: process.env.EMAIL_USER, // Use email from .env
    pass: process.env.EMAIL_PASS, // Use App Password from .env
```

```
  },
});

// OTP function for registration and password reset
const sendOtpEmail = async (email, otp, purpose = 'registration') => {
  const purposes = {
    registration: {
      subject: 'OTP for Registration',
      text: `Your OTP for registration is: ${otp}. It will expire in 10 minutes.`
    },
    reset: {
      subject: 'OTP for Password Reset',
      text: `Your OTP for password reset is: ${otp}. It will expire in 10 minutes.`
    }
  };

  if (!purposes[purpose]) {
    throw new Error('Invalid OTP purpose');
  }

  const mailOptions = {
    from: process.env.EMAIL_USER,
    to: email,
    subject: purposes[purpose].subject,
    text: purposes[purpose].text,
  };

  try {
    await transporter.sendMail(mailOptions);
    console.log(`☑ OTP email (${purpose}) sent successfully`);
  } catch (error) {
    console.error(`❌ Error sending OTP email (${purpose}):`, error);
    throw new Error(`Failed to send OTP for ${purpose}`);
  }
};

// General notification email function
const sendNotificationEmail = async (email, subject, message) => {
  const mailOptions = {
    from: process.env.EMAIL_USER,
    to: email,
    subject: subject,
    text: message,
  };

  try {
    await transporter.sendMail(mailOptions);
    console.log(`☑ Notification email sent to ${email}`);
  } catch (error) {
    console.error('❌ Error sending notification email:', error);
```

```
    throw new Error('Failed to send email notification');
  }
};

// Referral notification function
const sendReferralNotification = async (alumniEmail, studentName, position,
company, message, resumeUrl) => {
  const mailOptions = {
    from: process.env.EMAIL_USER,
    to: alumniEmail,
    subject: `New Referral Request for ${position} at ${company}`,
    text: `
      Dear Alumni,

      You have received a referral request from ${studentName} for:
      Position: ${position}
      Company: ${company}

      Student's Message:
      ${message}

      Resume: ${resumeUrl}

      Please log in to your account to respond to this request.

      Best regards,
      Alumni Connect Team
    `,
    html: `
      <div style="font-family: Arial, sans-serif; max-width: 600px; margin: 0 auto;
padding: 20px; border: 1px solid #e0e0e0; border-radius: 8px;">
        <h2 style="color: #2c3e50;">New Referral Request</h2>
        <p>Dear Alumni,</p>
        <p>You have received a referral request from
<strong>${studentName}</strong> for:</p>
        <div style="background-color: #f8f9fa; padding: 15px; border-radius: 5px;
margin: 15px 0;">
          <p style="margin: 5px 0;"><strong>Position:</strong> ${position}</p>
          <p style="margin: 5px 0;"><strong>Company:</strong> ${company}</p>
        </div>
        <div style="margin: 15px 0;">
          <p><strong>Student's Message:</strong></p>
          <p style="background-color: #f8f9fa; padding: 10px; border-radius:
5px;">${message}</p>
        </div>
        <div style="text-align: center; margin: 25px 0;">
          <a href="${resumeUrl}" target="_blank" style="
            display: inline-block;
            padding: 12px 24px;
            background-color: #4CAF50;
```

```
            color: white;
            text-decoration: none;
            border-radius: 4px;
            font-weight: bold;
          ">
            View Resume
          </a>
        </div>
        <p>Please log in to your account to respond to this request.</p>
        <div style="margin-top: 30px; padding-top: 15px; border-top: 1px solid
#e0e0e0;">
          <p>Best regards,<br>The Alumni Connect Team</p>
        </div>
      </div>
    `,
  };

  try {
    await transporter.sendMail(mailOptions);
    console.log('☑ Referral notification sent successfully');
  } catch (error) {
    console.error('❌ Error sending referral notification:', error);
    throw new Error('Failed to send referral notification');
  }
};

// Referral status update function
const sendReferralStatusUpdate = async (studentEmail, alumniName, position,
company, status, reason = '') => {
  const mailOptions = {
    from: process.env.EMAIL_USER,
    to: studentEmail,
    subject: `Update on Your Referral Request for ${position}`,
    text: `
      Dear Student,

      Your referral request for ${position} at ${company} has been
${status.toLowerCase()} by ${alumniName}.

      ${status === 'Rejected' ? `Reason: ${reason}` : ''}

      ${status === 'Accepted' ? 'Congratulations! The alumni has agreed to refer
you.' : ''}

      Best regards,
      Alumni Connect Team
    `,
  };

  try {
```

```
    await transporter.sendMail(mailOptions);
    console.log('☑ Referral status update sent successfully');
  } catch (error) {
    console.error('❌ Error sending status update:', error);
    throw new Error('Failed to send status update');
  }
};

module.exports = {
  sendOtpEmail,
  sendNotificationEmail,
  sendReferralNotification,
  sendReferralStatusUpdate
};
```

mentorshippage

```
import React, { useEffect, useState } from "react";
import axios from "axios";
import { FaTrash } from "react-icons/fa";
import "./MentorshipPage.css";

const MentorshipPage = () => {
    const [mentorship, setMentorship] = useState({});

    useEffect(() => {
        const fetchMentorship = async () => {
            try {
                const response = await
axios.get("http://localhost:5000/api/mentorship");
                setMentorship(response.data);
            } catch (error) {
                console.error("  Error fetching mentorship questions:", error);
            }
        };

        fetchMentorship();
    }, []);

    const handleDelete = async (questionId) => {
        if (window.confirm("Are you sure you want to delete this question?")) {
            try {
                console.log("Deleting question with ID:", questionId);
                const response = await
axios.delete(`http://localhost:5000/api/mentorship/${questionId}`);

                if (response.status === 200) {
                    setMentorship((prevMentorship) => {
```

```
                        const updatedMentorship = { ...prevMentorship };
                        Object.keys(updatedMentorship).forEach((studentName) => {
                            updatedMentorship[studentName] =
updatedMentorship[studentName].filter(
                                (question) => question._id !== questionId
                            );
                        });
                        return updatedMentorship;
                    });
                    alert("   Question deleted successfully");
                } else {
                    alert("  Failed to delete question.");
                }
            } catch (error) {
                console.error("  Error deleting question:", error);
                alert("  Error deleting question. Please try again.");
            }
        }
    };

    return (
        <div className="mentorship-page">
            <h2>Mentorship Questions by Students</h2>
            {Object.keys(mentorship).map((studentName) => (
                <div key={studentName} className="questions-by-student">
                    <h3>{studentName}</h3>
                    <ul>
                        {mentorship[studentName].map((question) => (
                            <li key={question._id}>
                                <h4>Question: {question.question}</h4>

                                {/*    Display Answers (Single or Multiple) */}
                                {question.answers && question.answers.length > 0 ?
(
                                    <div className="answers-container">
                                        {question.answers.map((ans, index) => (
                                            <p key={index}>
                                                <strong>{ans.alumniName}:</strong>
{ans.answer}
                                            </p>
                                        ))}
                                    </div>
                                ) : (
                                    <p style={{ fontStyle: "italic", color: "#666"
}}>No replies yet</p>
                                )}

                                <div className="question-actions">
                                    <button className="icon-button delete"
onClick={() =>
```

```
handleDelete(question._id)}>
                                        <FaTrash />
                                    </button>
                                </div>
                            </li>
                        ))}
                    </ul>
                </div>
            ))}
        </div>
    );
};

export default MentorshipPage;
```

postquestion

```
import React, { useState, useEffect } from "react";
import axios from "axios";
import "./PostQuestion.css";

const PostQuestion = () => {
  const [question, setQuestion] = useState("");
  const [questions, setQuestions] = useState([]);
  const [loading, setLoading] = useState(false);
  const token = localStorage.getItem("token");

  // Fetch User's Questions on Load
  useEffect(() => {
    const fetchQuestions = async () => {
      if (!token) return;

      try {
        const res = await
axios.get("http://localhost:5000/api/mentorship/my-questions", {
          headers: { Authorization: `Bearer ${token}` },
        });
        setQuestions(res.data);
      } catch (error) {
        console.error("Error fetching questions:", error.response?.data ||
error.message);
```

```
      }
    };

    fetchQuestions();
  }, [token]);

  // Handle Posting a Question
  const handleAskQuestion = async () => {
    if (!question.trim()) {
      alert("Please enter a question!");
      return;
    }

    if (!token) {
      console.error("No token found! User might not be logged in.");
      alert("You are not logged in! Please login first.");
      return;
    }

    setLoading(true);
    try {
      const res = await axios.post(
        "http://localhost:5000/api/mentorship/ask",
        { question },
        { headers: { Authorization: `Bearer ${token}` } }
      );

      alert("Question posted successfully!");

      if (res.data && res.data.question) {
        setQuestions((prevQuestions) => [...prevQuestions, res.data.question]);
      }

      setQuestion(""); // Clear input after success
    } catch (error) {
      console.error("Error posting question:", error.response?.data ||
error.message);
      alert("Failed to post question!");
    } finally {
      setLoading(false);
    }
  };

  return (
    <div className="post-question-container">
      <h2>Ask a Question</h2>
      <textarea
        className="question-textarea"
        placeholder="Type your question..."
        value={question}
```

```
        onChange={(e) => setQuestion(e.target.value)}
      />
      <button
        className="submit-btn"
        onClick={handleAskQuestion}
        disabled={loading}
      >
        {loading ? "Posting..." : "Post Question"}
      </button>

      <h2>Your Questions & Replies</h2>
      <div className="questions-list">
        {questions.length === 0 ? (
          <p>No questions asked yet.</p>
        ) : (
          questions.map((q) => (
            <div className="question-card" key={q._id}>
              <p><strong>You:</strong> {q.question}</p>

              {q.answers.length > 0 ? (
                <div className="answers-container">
                  {q.answers.map((ans, index) => (
                    <div className="answer-item" key={index}>
                      <p><strong>{ans.alumniName}:</strong> {ans.answer}</p>
                    </div>
                  ))}
                </div>
              ) : (
                <p className="no-replies">No replies yet</p>
              )}
            </div>
          ))
        )}
      </div>
    </div>
  );
};

export default PostQuestion;
```

viewquestion

```
import React, { useEffect, useState } from "react";
import axios from "axios";
import "./ViewQuestions.css";
```

```
const ViewQuestions = () => {
  const [questions, setQuestions] = useState([]);
  const [reply, setReply] = useState({});
  const [refreshing, setRefreshing] = useState(false);

  const fetchQuestions = async () => {
    try {
      const token = localStorage.getItem("token");
      if (!token) {
        console.error("No token found! Login required.");
        return;
      }

      const res = await axios.get("http://localhost:5000/api/mentorship/questions",
{
        headers: { Authorization: `Bearer ${token}` },
      });
      setQuestions(res.data);
    } catch (error) {
      console.error("Error fetching questions:", error);
    } finally {
      setRefreshing(false);
    }
  };

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

  const handleReply = async (id) => {
    try {
      const token = localStorage.getItem("token");
      if (!token) return;

      setRefreshing(true);
      await axios.post(
        `http://localhost:5000/api/mentorship/reply/${id}`,
        { answer: reply[id] },
        { headers: { Authorization: `Bearer ${token}` } }
      );
      await fetchQuestions();
      setReply((prev) => ({ ...prev, [id]: "" }));
    } catch (error) {
      console.error("Error sending reply:", error);
      setRefreshing(false);
    }
  };

  const handleIgnore = async (id) => {
```

```
  try {
    const token = localStorage.getItem("token");
    if (!token) return;

    setRefreshing(true);
    await axios.post(
      `http://localhost:5000/api/mentorship/ignore/${id}`,
      {},
      { headers: { Authorization: `Bearer ${token}` } }
    );
    setQuestions(questions.filter(q => q._id !== id));
  } catch (error) {
    console.error("Error ignoring question:", error);
    setRefreshing(false);
  }
};

return (
  <div className="view-questions-container">
    <h2 className="view-questions-title">Student Questions</h2>
    {refreshing && <div className="updating-indicator">Updating...</div>}

    {questions.length === 0 ? (
      <div className="empty-state">
        <p>No unanswered questions found</p>
      </div>
    ) : (
      <div className="questions-list">
        {questions.map((q, index) => (
          <div
            key={q._id}
            className="question-card"
            style={{ "--index": index }}
          >
            <div className="question-header">
              <p className="student-name">{q.studentName}:</p>
              <p className="question-text">{q.question}</p>
            </div>

            <div className="reply-form">
              <textarea
                placeholder="Type your reply..."
                value={reply[q._id] || ""}
                onChange={(e) => setReply({ ...reply, [q._id]: e.target.value })}
                className="reply-textarea"
                rows="3"
              />
              <div className="button-group">
                <button
                  onClick={() => handleReply(q._id)}
```

```
                  disabled={!reply[q._id] || refreshing}
                  className="button reply-button"
                >
                  Reply
                </button>
                <button
                  onClick={() => handleIgnore(q._id)}
                  disabled={refreshing}
                  className="button ignore-button"
                >
                  Ignore
                </button>
              </div>
            </div>
          </div>
        ))}
      </div>
    )}
  </div>
);
};

export default ViewQuestions;
```

mentorshipcontroller

```
const Question = require('../models/Question'); // Import the Question model

exports.postQuestion = async (req, res) => {
    try {
        const { question, userId } = req.body;

        // Ensure question and userId are provided
        if (!question || !userId) {
            return res.status(400).json({ error: "Question and userId are required"
});
        }

        // Create and save the question
        const newQuestion = new Question({ question, userId });
        await newQuestion.save();

        res.status(201).json({ message: "Question posted successfully!", question:
```

```
newQuestion });
    } catch (error) {
        console.error("Error posting question:", error);
        res.status(500).json({ error: "Internal Server Error" });
    }
};
```

Questions

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

const QuestionSchema = new mongoose.Schema({
  studentId: { type: mongoose.Schema.Types.ObjectId, ref: "User", required: true },

  studentName: { type: String, required: true },
  question: { type: String, required: true },
  answers: [
    {
      alumniId: { type: mongoose.Schema.Types.ObjectId, ref: "User" },
      alumniName: { type: String },
      answer: { type: String, required: true },
      answeredAt: { type: Date, default: Date.now }
    }
  ],
  ignoredBy: [{ type: mongoose.Schema.Types.ObjectId, ref: "User" }], // Alumni who
ignored
  createdAt: { type: Date, default: Date.now }
});

module.exports = mongoose.model("Question", QuestionSchema);
```

routes/mentorship

```
const express = require("express");
const router = express.Router();
const Question = require("../models/Question");
const verifyToken = require("../middleware/auth");
const User = require("../models/User");

//   Student: Ask a Question
router.post("/ask", verifyToken, async (req, res) => {
```

```
  try {
    console.log("    Received request:", req.body);
    console.log("    User from token:", req.user);

    const { question } = req.body;

    if (!question) {
      return res.status(400).json({ message: "Question is required" });
    }

    if (!req.user || !req.user.id || !req.user.name) {
      return res.status(401).json({ message: "Unauthorized. User details missing."
});
    }

    const newQuestion = new Question({
      studentId: req.user.id,
      studentName: req.user.name,
      question,
    });

    await newQuestion.save();
    console.log("   Question saved:", newQuestion);

    res.status(201).json({ message: "Question posted successfully!", question:
newQuestion });
  } catch (error) {
    console.error("  Error in /ask route:", error);
    res.status(500).json({ message: "Internal Server Error", error: error.message
});
  }
});

//   Alumni: Get All Unanswered Questions (Updated)
router.get("/questions", verifyToken, async (req, res) => {
  try {
    if (req.user.role !== "Alumni") {
      return res.status(403).json({ message: "Access Denied" });
    }

    const questions = await Question.find({
      "answers.alumniId": { $ne: req.user.id }, // Exclude if alumni already
answered
      ignoredBy: { $ne: req.user.id } // Exclude if alumni ignored
    })
    .select("studentName question answers")
    .lean();

    res.json(questions);
  } catch (error) {
```

```
    console.error("  Error fetching questions:", error);
    res.status(500).json({ message: "Server Error", error: error.message });
  }
});

//   Alumni: Reply to a Question (Updated)
router.post("/reply/:id", verifyToken, async (req, res) => {
  try {
    if (req.user.role !== "Alumni") {
      return res.status(403).json({ message: "Access Denied" });
    }

    const { id } = req.params;
    const { answer } = req.body;

    const alumni = await User.findById(req.user.id).select("name");
    if (!alumni) {
      return res.status(404).json({ message: "Alumni not found" });
    }

    const question = await Question.findById(id);
    if (!question) return res.status(404).json({ message: "Question not found" });

    // Append new answer instead of replacing
    question.answers.push({
      alumniId: req.user.id,
      alumniName: alumni.name,
      answer
    });

    await question.save();

    res.json({
      message: "Reply sent successfully!",
      question
    });
  } catch (error) {
    console.error("  Error in /reply route:", error);
    res.status(500).json({ message: "Server Error", error: error.message });
  }
});

//      Alumni: Ignore a Question
router.post("/ignore/:id", verifyToken, async (req, res) => {
  try {
    if (req.user.role !== "Alumni") {
      return res.status(403).json({ message: "Access Denied" });
    }

    const question = await Question.findByIdAndUpdate(
```

```
      req.params.id,
      { $addToSet: { ignoredBy: req.user.id } }, // Add to ignoredBy array
      { new: true }
    );

    if (!question) {
      return res.status(404).json({ message: "Question not found" });
    }

    res.json({
      message: "Question ignored successfully",
      questionId: question._id
    });
  } catch (error) {
    console.error("  Error in /ignore route:", error);
    res.status(500).json({ message: "Server Error", error: error.message });
  }
});

//   Student: Get Their Own Questions and Answers
router.get("/my-questions", verifyToken, async (req, res) => {
  try {
    const questions = await Question.find({ studentId: req.user.id })
      .select("studentName question answers createdAt")
      .sort({ createdAt: -1 });

    res.json(questions);
  } catch (error) {
    console.error("  Error in /my-questions route:", error);
    res.status(500).json({ message: "Server Error", error: error.message });
  }
});

module.exports = router;
```

server.js

```
const express = require("express");
```

```
const mongoose = require("mongoose");
const cors = require("cors");
const path = require("path");
require("dotenv").config();

// Import Routes
const adminRoutes = require("./routes/adminRoutes");
const adminUserRoutes = require("./routes/adminUsers");
const adminreportRoutes = require("./routes/adminReports");
const dashboardRoutes = require("./routes/dashboardRoutes");
const admininternworkRoutes = require("./routes/admininternworkroutes");
const adminmentorprojRoutes = require("./routes/adminmentorprojroutes");


const adminresourceRoutes = require("./routes/adminresources");


const uploadRoutes = require("./routes/upload");
const registerRoute = require("./routes/register");
const loginRoute = require("./routes/login");
const resourceRoutes = require("./routes/resources");
const downloadRoutes = require("./routes/download");
const reviewRoutes = require("./routes/reviewRoutes");
const internshipRoutes = require('./routes/internshipRoutes');
const workshopRoutes = require("./routes/workshop");
const bookmarkRoutes = require("./routes/bookmarkRoutes");
const authRoutes = require("./routes/auth");
const reportRoutes = require("./routes/reportRoutes");
const fundRoutes = require("./routes/FundRoutes");
const mentorshipRoutes = require("./routes/mentorship");
const userRoutes = require("./routes/userRoutes");
const projectRoutes = require("./routes/projectRoutes");
const referralRoutes = require("./routes/referralRoutes");
const profileRoutes = require('./routes/profileRoutes');
const authRoutess = require('./routes/authRoutes');




const app = express();

// Debug log to check if the server receives requests
app.use((req, res, next) => {
    console.log(`📩 Request Received: ${req.method} ${req.url}`);
    console.log("🔹 Body:", req.body);
    next();
});
```

```
// Middleware
app.use(express.json());
app.use(express.urlencoded({ extended: true }));
app.use(cors({ origin: "http://localhost:3000", credentials: true }));

// Serve static files
app.use("/uploads", express.static(path.join(__dirname, "uploads")));
app.use('/referral-resumes', express.static(path.join(__dirname,
'referral_resumes')));
app.use('/uploads/profile-pictures', express.static(path.join(__dirname,
'uploads/profile-pictures')));

// Use Routes
app.use("/api/upload", uploadRoutes);
app.use("/api", registerRoute);
app.use("/api", loginRoute);
app.use("/api", adminRoutes);
app.use("/api/admin", adminUserRoutes);
app.use("/api/reports", adminreportRoutes);
app.use("/api/resources", resourceRoutes);
app.use("/", downloadRoutes);
app.use("/api/reviews", reviewRoutes);
app.use('/api/internships', internshipRoutes);
app.use("/api/workshops", workshopRoutes);
app.use("/api/bookmarks", bookmarkRoutes);
app.use("/api", authRoutes);
app.use("/api/reports", reportRoutes);
app.use("/api", fundRoutes);
app.use("/api/mentorship", mentorshipRoutes);
app.use("/api/user", userRoutes);
app.use("/api", projectRoutes);
app.use("/api/referrals", referralRoutes);
app.use('/api/profile', profileRoutes);
app.use('/api', authRoutess); // This prefixes all authRoutes with '/api'

app.use("/api", admininternworkRoutes);
app.use("/api", adminmentorprojRoutes);
app.use("/", dashboardRoutes);

app.use("/api/admin/resources", adminresourceRoutes);
```

```
// Ensure uploads/ and subdirectories exist
const fs = require("fs");
const uploadDir = path.join(__dirname, "uploads");
const profilePicturesDir = path.join(uploadDir, "profile-pictures");

if (!fs.existsSync(uploadDir)) {
    fs.mkdirSync(uploadDir, { recursive: true });
    console.log("☑ Created 'uploads' folder");
}

if (!fs.existsSync(profilePicturesDir)) {
    fs.mkdirSync(profilePicturesDir, { recursive: true });
    console.log("☑ Created 'uploads/profile-pictures' folder");
}

// Connect to MongoDB
mongoose.connect("mongodb://127.0.0.1:27017/alumnilink", {})
    .then(() => console.log("☑ MongoDB Connected"))
    .catch(err => console.error("❌ MongoDB connection error:", err));

// Start Server
const PORT = process.env.PORT || 5000;
app.listen(PORT, () => console.log(`🚀 Server running on port ${PORT}`));
```

app.js

```
import React, { useState, useEffect } from "react";
import { BrowserRouter as Router, Route, Routes, useLocation } from
"react-router-dom";
import FrontPage from "./FrontPage";
import Login from "./login";
import Register from "./Register";
import Dashboard from "./Dashboard";
import AdminLogin from "./AdminLogin";
import AdminDashboard from "./AdminDashboard";

import InternshipsPage from './AdminInternships';  // Page to display internships
import WorkshopsPage from './AdminWorkshops';  // Page to display workshops
import ProjectsPage from "./ProjectsPage";
import MentorshipPage from "./MentorshipPage";

import AdminResources from "./AdminResources";
```

```
import Resources from "./Resources";
import AlumniReview from "./AlumnilinkReview";
import StudentReview from "./StudentReview";
//import AlumniProfile from "./AlumniProfile"; // Import Alumni Profile Page
//import StudentProfile from "./StudentProfile";
// //import EditAlumniProfile from "./EditAlumniProfile";
import AddInternship from "./AddInternship";
import ViewInternships from "./ViewInternships";
import AddWorkshop from "./AddWorkshop";
import ViewWorkshops from "./ViewWorkshops";
import WorkshopDetails from "./WorkshopDetails";
import InternshipDetails from "./InternshipDetails";
import Bookmarks from "./Bookmarks";
import Reports from "./Reports";
//import FundDetails from "./FundDetails";
//import StudentFund from "./StudentFund";
import StudentFund from "./StudentFund";
import AdminFund from "./AdminFund";
import AlumniFund from "./AlumniFund";

import PostQuestions from "./PostQuestion";
import ViewQuestions from "./ViewQuestions";
import AddUser from "./AddUser";
import AddReport from "./AddReport";
// import ProfileView from "./ProfileView"; // Import the profile view page
// import EditProfile from "./EditProfile";
import AlumniPage from "./AlumniPage";
import StudentPage from "./StudentPage";
import ProjectApplications from "./ProjectApplications";

import ReferralRequestForm from "./ReferralRequestForm";
import AlumniReferralList from "./AlumniReferralList";
import StudentReferralList from "./StudentReferralList";
import Profile from "./Profile";
import EditProfile from './EditProfile';
import ForgotPassword from "./ForgotPassword";

import "./App.css";

function App() {
  return (
    <Router>
      <AppWrapper />
    </Router>
  );
}

function AppWrapper() {
  const location = useLocation();
```

```
  const isAuthPage = location.pathname === "/" || location.pathname ===
"/register";
  const [userType, setUserType] = useState(localStorage.getItem("userType") ??
"student");

  useEffect(() => {
    const storedUserType = localStorage.getItem("userType") ?? "student";
    console.log("Stored userType in localStorage:", storedUserType);
    setUserType(storedUserType);
  }, []);

  console.log("Current userType:", userType);

  return (
    <div className={isAuthPage ? "auth-background" : "dashboard-background"}>
      <Routes>
        {/* Public Routes */}
        <Route path="/" element={<FrontPage />} />
        <Route path="/login" element={<Login />} />
        <Route path="/register" element={<Register />} />
        <Route path="/admin-login" element={<AdminLogin />} />
        <Route path="/admin-dashboard" element={<AdminDashboard />} />
        <Route path="/add-user" element={<AddUser />} />
        <Route path="/add-report" element={<AddReport />} />
        {/* Protected Routes */}
        <Route path="/dashboard" element={<Dashboard />} />
        <Route path="/resources" element={<Resources />} />

        {/*<Route path="/alumni/profile" element={<AlumniProfile />} />
        <Route path="/student/profile" element={<StudentProfile />} />
        <Route path="/edit-profile" element={<EditAlumniProfile />} />*/}

        <Route path="/add-internship" element={<AddInternship />} />
        <Route path="/view-internship" element={<ViewInternships />} />
        <Route path="/add-workshop" element={<AddWorkshop />} />
        <Route path="/view-workshops" element={<ViewWorkshops />} />
        <Route path="/internship/:id" element={<InternshipDetails />} />
        <Route path="/workshop/:id" element={<WorkshopDetails />} />
        {/* Review Pages */}
        <Route path="/AlumniReview" element={<AlumniReview />} />
        <Route path="/StudentReview" element={<StudentReview />} />
        <Route path="/bookmarks" element={<Bookmarks />} />
        <Route path="/reports" element={<Reports />} />
        {/*<Route path="/fund-details" element={<FundDetails />} />*/}
        <Route path="/student" element={<StudentFund />} />
        <Route path="/admin-fund" element={<AdminFund />} />
        <Route path="/alumni" element={<AlumniFund />} />
        <Route path="/post-questions" element={<PostQuestions />} />
        <Route path="/view-questions" element={<ViewQuestions />} />
        {/* <Route path="/profile" element={<ProfileView />} /> */}
```

```
        {/* <Route path="/edit-profile" element={<EditProfile />} /> */}

        <Route path="/profile/:userId" element={<Profile />} />
        <Route path="/profile" element={<Profile />} />

        <Route path="/edit-profile/:userId" element={<EditProfile />} />

        <Route path="/admin/resources" element={<AdminResources />} />


        <Route path="/alumni-page" element={<AlumniPage />} />
        <Route path="/student-page" element={<StudentPage />} />
        <Route path="alumni-page/project/:projectId" element={<ProjectApplications
/>} />

       {/* New Referral System Routes */}
{/* <Route path="/request-referral/:internshipId/:alumniId"
element={<ReferralRequestForm />} /> */}

<Route path="/request-referral/:internshipId" element={<ReferralRequestForm />} />
<Route path="/alumni/referrals" element={<AlumniReferralList />} />
<Route path="/student/referrals" element={<StudentReferralList />} />
<Route path="/forgot-password" element={<ForgotPassword />} />

<Route path="/internships" element={<InternshipsPage />} />
        <Route path="/workshops" element={<WorkshopsPage />} />
        <Route path="/projects" element={<ProjectsPage />} />
        {/* <Route path="/forgot-password" element={<ForgotPassword />} /> */}
        <Route path="/mentorship" element={<MentorshipPage />} />
        <Route path="*" element={<h1>404 - Page Not Found</h1>} />
      </Routes>
    </div>
  );
}

export default App;
```