**Login.js**

```
import React, { useState } from "react";
import { useNavigate, Link } from "react-router-dom";
import axios from "axios"; // Use axios for API calls
import "./style.css";
import heroImage from "./hero-image2.jpg";

const Login = () => {
  const [formData, setFormData] = useState({
    email: "",
    password: "",
    role: "Student", // Default role
  });

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

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

  const handleSubmit = async (e) => {
    e.preventDefault();
    setError("");

    console.log("Sending login request:", formData);

    try {
      const response = await axios.post("http://localhost:5000/api/login", formData);
      const { token, role, name } = response.data; // Assuming the backend sends the user's name

      console.log("Login successful! Token:", token); // Debugging: Log the token

      // Store token, role, and name in localStorage
      localStorage.setItem("token", token);
      localStorage.setItem("userRole", role);
      localStorage.setItem("userName", name); // Store the user's name

      alert("Login successful!");

      // Redirect based on role
      if (role === "Alumni") {
        navigate("/dashboard"); // Redirect to alumni dashboard
      } else if (role === "Student") {
        navigate("/dashboard"); // Redirect to student dashboard
      } else {
        navigate("/dashboard"); // Fallback for unknown roles
      }
    } catch (error) {
      console.error("Login error:", error.response?.data?.message || error.message);
    setError(error.response?.data?.message || "Something went wrong. Please try again.");
  }
};

return (
  <div>
    {/* Navbar */}
    <nav className="navbar">
      <div className="logo">ALUMNILINK</div>
      <ul>
        <li><Link to="/">Home</Link></li>
        <li><Link to="/services">Services</Link></li>
        <li><Link to="/contact">Contact</Link></li>
        <li><Link to="/about">About</Link></li>
      </ul>
    </nav>
    <div className="login-image"></div>

          <div className="auth-section">
          <div className="auth-image">
            <img src={heroImage} alt="Handshake" />
          </div>

    {/* Login Container */}
    <div className="login-container">
      <h2>Login</h2>
      {error && <p className="error-message">{error}</p>}

      <form onSubmit={handleSubmit}>
        <input
          type="email"
          name="email"
          placeholder="Email"
          value={formData.email}
          onChange={handleChange}
          required
        />
        <input
          type="password"
          name="password"
          placeholder="Password"
          value={formData.password}
          onChange={handleChange}
          required
        />

        {/* Role Selection */}
        <select name="role" value={formData.role} onChange={handleChange} required>
          <option value="Student">Student</option>
          <option value="Alumni">Alumni</option>
        </select>
```

```
          {/* Forgot Password */}
          <p className="forgot-password" onClick={() => navigate("/forgot-password")}>
            Forgot Password?
          </p>

          <button type="submit">Login</button>
        </form>

        <p>Don't have an account? <Link to="/register">Register Here</Link></p>
      </div>
    </div>
    </div>
  );
};

export default Login;
```

**Register.js**

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

const Register = () => {
  const [step, setStep] = useState(1);
  const [otpVerified, setOtpVerified] = useState(false); // Ensure OTP is verified before registration
  const [formData, setFormData] = useState({
    admissionNumber: "",
    name: "",
    email: "",
    department: "",
    role: "Student",
    password: "",
    confirmPassword: "",
  });

  const [additionalData, setAdditionalData] = useState({});
  const [otp, setOtp] = useState("");
  const [error, setError] = useState("");
  const [files, setFiles] = useState({
    profilePicture: null,
    idCard: null,
    idProof: null,
  });
  const navigate = useNavigate();

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

  const handleAdditionalChange = (e) => {
    setAdditionalData({ ...additionalData, [e.target.name]: e.target.value });
  };

  const handleFileChange = (e) => {
    const { name, files } = e.target;
    setFiles((prev) => ({ ...prev, [name]: files[0] }));
  };

  const sendOtp = async () => {
    try {
      const response = await axios.post("http://localhost:5000/api/send-otp", {
        admissionNumber: formData.admissionNumber,
        name: formData.name,
        email: formData.email,
      });
      alert(response.data.message);
      setStep(2);
    } catch (error) {
      setError(error.response?.data?.message || "Error sending OTP.");
    }
  };

  const verifyOtp = async () => {
    try {
      const response = await axios.post("http://localhost:5000/api/verify-otp", {
        email: formData.email,
        otp,
      });
      if (response.data.success) {
        setOtpVerified(true);
        setStep(3);
      } else {
        setError("Invalid OTP");
      }
    } catch (error) {
      setError(error.response?.data?.message || "OTP verification failed.");
    }
  };

  const handleSubmit = async (e) => {
    e.preventDefault();
    setError("");

    if (!otpVerified) {
      setError("OTP not verified. Please verify before registration.");
      return;
    }

    if (formData.password !== formData.confirmPassword) {
      setError("Passwords do not match.");
      return;
    }

    const formDataWithFiles = new FormData();
    for (const key in formData) {
      formDataWithFiles.append(key, formData[key]);
    }
    for (const key in additionalData) {
      formDataWithFiles.append(key, additionalData[key]);
    }
    if (files.idCard) formDataWithFiles.append("idCard", files.idCard);
    if (files.profilePicture) formDataWithFiles.append("profilePicture", files.profilePicture);
    if (files.idProof) formDataWithFiles.append("idProof", files.idProof);

    try {
      await axios.post("http://localhost:5000/api/register", formDataWithFiles);
      alert("Registration successful! You can now log in.");
      navigate("/login");
    } catch (error) {
      setError(error.response?.data?.message || "Something went wrong.");
    }
  };

  return (
    <div>
      <h2>Register</h2>
      {error && <p className="error-message">{error}</p>}

      {step === 1 && (
        <form onSubmit={(e) => { e.preventDefault(); sendOtp(); }}>
          <input type="text" name="admissionNumber" placeholder="Admission Number" required
onChange={handleChange} />
          <input type="text" name="name" placeholder="Name" required onChange={handleChange} />
          <input type="email" name="email" placeholder="Email" required onChange={handleChange} />
          <input type="text" name="department" placeholder="Department" required
onChange={handleChange} />
          <select name="role" value={formData.role} onChange={handleChange} required>
            <option value="Student">Student</option>
            <option value="Alumni">Alumni</option>
          </select>
          <button type="submit">Send OTP</button>
        </form>
      )}

      {step === 2 && (
        <form onSubmit={(e) => { e.preventDefault(); verifyOtp(); }}>
          <input type="text" placeholder="Enter OTP" value={otp} onChange={(e) =>
setOtp(e.target.value)} required />
          <button type="submit">Verify OTP</button>
        </form>
      )}

      {step === 3 && (
        <form onSubmit={handleSubmit}>
          <input type="password" name="password" placeholder="Create Password" required
onChange={handleChange} />
          <input type="password" name="confirmPassword" placeholder="Confirm Password" required
onChange={handleChange} />

          <input type="text" name="gender" placeholder="Gender" required
onChange={handleAdditionalChange} />
          <input type="date" name="dob" placeholder="Date of Birth" required
onChange={handleAdditionalChange} />

          {formData.role === "Alumni" && (
            <>
              <input type="text" name="currentJobTitle" placeholder="Current Job Title" required
onChange={handleAdditionalChange} />
```

```
              <input type="text" name="companyName" placeholder="Company Name" required
onChange={handleAdditionalChange} />
              <input type="text" name="industry" placeholder="Industry" required
onChange={handleAdditionalChange} />
              <input type="number" name="yearsOfExperience" placeholder="Years of Experience" required
onChange={handleAdditionalChange} />
              <input type="number" name="graduationYear" placeholder="Graduation Year" required
onChange={handleAdditionalChange} />
              <input type="file" name="idProof" onChange={handleFileChange} required />
            </>
          )}

          {formData.role === "Student" && (
            <>
              <input type="text" name="ktuId" placeholder="KTU ID" required
onChange={handleAdditionalChange} />
              <input type="number" name="yearOfAdmission" placeholder="Year of Admission" required
onChange={handleAdditionalChange} />
              <input type="number" name="currentYear" placeholder="Current Year of Study" required
onChange={handleAdditionalChange} />
              <input type="text" name="rollNumber" placeholder="Roll Number" required
onChange={handleAdditionalChange} />
              <input type="number" name="graduationYear" placeholder="Graduation Year" required
onChange={handleAdditionalChange} />
              <input type="file" name="idCard" onChange={handleFileChange} required />
            </>
          )}

          <input type="file" name="profilePicture" onChange={handleFileChange} required />
          <button type="submit">Complete Registration</button>
        </form>
      )}
    </div>
  );
};

export default Register;
```

**models/AdminUserModel.js**

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

const adminUserSchema = new mongoose.Schema({
  admissionNumber: { type: String, required: true, unique: true },
  name: { type: String, required: true },
  email: { type: String, required: true, unique: true } // Check if email is included
});

const AdminUser = mongoose.model('AdminUser', adminUserSchema);
module.exports = AdminUser;
```

**models/AlumniModel.js**

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

const alumniSchema = new mongoose.Schema({
  userId: { type: mongoose.Schema.Types.ObjectId, ref: "User", required: true },
  admissionNumber: { type: String, required: true },
  name: { type: String, required: true },
  email: { type: String, required: true },
  gender: { type: String, required: true },
  dob: { type: Date, required: true },
  department: { type: String, required: true },
  graduationYear: { type: Number, required: true },
  profilePicture: { type: String },
  idProof: { type: String },
  industry: { type: String },
  yearsOfExperience: { type: Number },
  currentJobTitle: { type: String },
  company: { type: String },
  linkedInProfile: { type: String },
  isVerified: { type: Boolean, default: false },
});

module.exports = mongoose.model('Alumni', alumniSchema);
```

**models/OtpModel.js**

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

const otpSchema = new mongoose.Schema({
  email: { type: String, required: true },
  otp: { type: String, required: true },
  createdAt: { type: Date, default: Date.now, expires: 300 }, // OTP expires in 5 minutes
  verified: { type: Boolean, default: false }  // ✅ Add this line
});
```

```
module.exports = mongoose.model("OTP", otpSchema);
```

**models/StudentModel.js**

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

const studentSchema = new mongoose.Schema({
  userId: { type: mongoose.Schema.Types.ObjectId, ref: "User", required: true },
  admissionNumber: { type: String, required: true },
  name: { type: String, required: true },
  email: { type: String, required: true },
  gender: { type: String, required: true },
  dob: { type: Date, required: true },
  department: { type: String, required: true },
  yearOfAdmission: { type: Number, required: true },
  currentYear: { type: String, required: true }, // Ensure this field is sent from frontend
  rollNumber: { type: String, required: true },
  ktuId: { type: String, required: true },  // ✅ Added KTU ID
  graduationYear: { type: Number, required: true },  // ✅ Added Graduation Year
  idCard: { type: String },
  profilePicture: { type: String },
  isVerified: { type: Boolean, default: false },
});

module.exports = mongoose.model('Student', studentSchema);
```

**models/User.js**

```
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, trim: true },
    password: { type: String, required: true, minlength: 6 },
    role: { type: String, enum: ["Student", "Alumni"], required: true }
}, { timestamps: true });

// Hash password before saving (This is already correct)
userSchema.pre("save", async function (next) {
    next();
});

const User = mongoose.model("User", userSchema);
module.exports = User;
```

routes/login.js

**routes/login.js**

```
const express = require("express");
const bcrypt = require("bcryptjs");
const jwt = require("jsonwebtoken");
const User = require("../models/User"); // Import User model
const router = express.Router();
require("dotenv").config(); // Load environment variables

// Login Route
//const bcrypt = require("bcryptjs"); // Ensure bcrypt is imported

router.post("/login", async (req, res) => {
  try {
    const { email, password } = req.body;

    console.log("📌 Login Attempt for Email:", email);
    console.log("📌 Entered Password:", password);

    const user = await User.findOne({ email });
    if (!user) {
      console.log("❌ User Not Found");
      return res.status(400).json({ message: "Invalid email or password" });
    }

    console.log("✅ User Found in DB:", user);
    console.log("📌 Stored Hashed Password:", user.password);

    // Check if bcrypt.compare() is working correctly
    bcrypt.compare(password, user.password, (err, isMatch) => {
      if (err) {
        console.error("❌ bcrypt.compare() Error:", err);
        return res.status(500).json({ message: "Server error" });
      }
      console.log("📌 bcrypt.compare() Result:", isMatch);

      if (!isMatch) {
        console.log("❌ Password Mismatch!");
        return res.status(400).json({ message: "Invalid email or password" });
      }

      console.log("✅ Password Matched!");
      res.status(200).json({ message: "Login successful" });
    });

  } catch (error) {
    console.error("❌ Login Error:", error);
    res.status(500).json({ message: "Server error" });
  }
});
```

```
module.exports = router;
```

**routes/register.js**

```
const express = require("express");
const multer = require("multer");
const path = require("path");
const bcrypt = require("bcryptjs");
const Alumni = require("../models/AlumniModel");
const Student = require("../models/StudentModel");
const User = require("../models/User");
const { sendOtpEmail } = require("../utils/email");
const OTP = require("../models/OtpModel");
const AdminUser = require("../models/AdminUserModel");

const router = express.Router();

// Configure multer for file uploads
const storage = multer.diskStorage({
  destination: (req, file, cb) => {
    cb(null, "uploads/");
  },
  filename: (req, file, cb) => {
    cb(null, Date.now() + path.extname(file.originalname));
  },
});
const upload = multer({ storage });

// Step 1: Check if User Exists & Send OTP
router.post("/send-otp", async (req, res) => {
  try {
    const { admissionNumber, name, email } = req.body;

    // Check if user exists in Admin Database
    const adminUser = await AdminUser.findOne({ admissionNumber, name });
    if (!adminUser) {
      return res.status(400).json({ message: "User not found in records. Please contact admin." });
    }

    // Generate & Store OTP
    const otpCode = Math.floor(100000 + Math.random() * 900000);
    const otpData = {
      email,
      otp: otpCode,
      createdAt: Date.now(),
      verified: false,
    };

    await OTP.updateOne({ email }, otpData, { upsert: true });
    await sendOtpEmail(email, otpCode);
    res.status(200).json({ message: "OTP sent successfully. Please verify." });
  } catch (error) {
    res.status(500).json({ message: "Failed to send OTP.", error: error.message });
  }
});

// Step 2: Verify OTP
router.post("/verify-otp", async (req, res) => {
  try {
    const { email, otp } = req.body;
    const otpRecord = await OTP.findOne({ email });
    console.log("OTP Record found:", otpRecord);

    if (!otpRecord || otpRecord.otp.toString() !== otp.toString()) {
      return res.status(400).json({ message: "Invalid OTP. Please try again." });
    }

    const otpExpiryTime = 10 * 60 * 1000;
    if (Date.now() - otpRecord.createdAt > otpExpiryTime) {
      return res.status(400).json({ message: "OTP has expired. Please request a new one." });
    }

    // Mark OTP as verified
    await OTP.updateOne({ email }, { $set: { verified: true } });
    console.log("OTP Verified for:", email);

    res.status(200).json({ success: true, message: "OTP verified successfully." });
  } catch (error) {
    res.status(500).json({ message: "Failed to verify OTP.", error: error.message });
  }
});

// Step 3: Register User (Without OTP Validation Again)
router.post(
  "/register",
  upload.fields([
    { name: "idCard", maxCount: 1 },
    { name: "profilePicture", maxCount: 1 },
    { name: "idProof", maxCount: 1 },
  ]),
  async (req, res) => {
    try {
      console.log("📌 Received Registration Request:", req.body);

      const {
        admissionNumber,
        name,
        email,
        password,
        gender,
        dob,
        role,
        department,
        yearOfAdmission,
        graduationYear,
        currentYear,
        rollNumber,
        ktuId,
        industry,
        yearsOfExperience,
        currentJobTitle,
        company,
      } = req.body;

      if (!email || !password || !role || !name || !admissionNumber) {
        return res.status(400).json({ message: "Required fields missing" });
      }

      // Check if OTP was verified before allowing registration
      const otpRecord = await OTP.findOne({ email });
      if (!otpRecord || !otpRecord.verified) {
        return res.status(400).json({ message: "OTP not verified. Please verify before registration." });
      }

      const salt = await bcrypt.genSalt(10);
      // ✅ FIX: Ensure `hashedPassword` is declared **before** using it
      const hashedPassword = await bcrypt.hash(password, salt);

      console.log("Original Password:", password);
      console.log("Hashed Password Stored:", hashedPassword);

      // Create User in `User` collection
      const user = new User({
        admissionNumber,
        name,
        email,
        password: hashedPassword, // ✅ Correctly placed now
        role,
      });
      await user.save();
      console.log("✅ User saved successfully:", user);

      if (role.toLowerCase() === "student") {
        console.log("📌 Saving student data...");
        const student = new Student({
          userId: user._id,
          admissionNumber,
          name,
          email,
          gender,
          dob,
          department,
          yearOfAdmission: parseInt(yearOfAdmission),
          graduationYear: parseInt(graduationYear),
          ktuId,
          currentYear,
          rollNumber,
          idCard: req.files["idCard"] ? req.files["idCard"][0].path : null,
          profilePicture: req.files["profilePicture"] ? req.files["profilePicture"][0].path : null,
          isVerified: false,
        });
        await student.save();
        console.log("✅ Student data saved successfully:", student);
      } else if (role.toLowerCase() === "alumni") {
        console.log("📌 Saving alumni data...");
        const alumni = new Alumni({
          userId: user._id,
          admissionNumber,
          name,
          email,
          gender,
          dob,
          department,
          graduationYear: parseInt(graduationYear),
          profilePicture: req.files["profilePicture"] ? req.files["profilePicture"][0].path : null,
          idProof: req.files["idProof"] ? req.files["idProof"][0].path : null,
          industry,
          yearsOfExperience: parseInt(yearsOfExperience),
          currentJobTitle,
          company,
          isVerified: false,
        });
        await alumni.save();
        console.log("✅ Alumni data saved successfully:", alumni);
      } else {
        console.log("❌ Invalid role:", role);
        return res.status(400).json({ message: "Invalid role provided" });
      }

      // Delete OTP after successful registration
      await OTP.deleteOne({ email });

      res.status(200).json({ message: "Registration successful. You can now log in." });
     } catch (error) {
      console.error("❌ Registration failed:", error);
      res.status(500).json({ message: "Registration failed.", error: error.message });
     }
    }
```

```
);

module.exports = router;
```

**Routes/uploads.js**

```
const express = require("express");
const multer = require("multer");
const path = require("path");
const Resource = require("../models/Resource");

const router = express.Router();

// ✅ Ensure uploads/ folder exists
const fs = require("fs");
const uploadDir = path.join(__dirname, "../uploads");
if (!fs.existsSync(uploadDir)) {
  fs.mkdirSync(uploadDir, { recursive: true });
  console.log("✅ Created 'uploads' folder");
}

// Set storage engine
const storage = multer.diskStorage({
  destination: (req, file, cb) => {
    const uploadPath = path.join(__dirname, "../uploads"); // Ensure correct folder
    console.log("Upload Path:", uploadPath); // Debugging log
    cb(null, uploadPath);
  },
  filename: (req, file, cb) => {
    cb(null, file.fieldname + "-" + Date.now() + path.extname(file.originalname));
  }
});

const upload = multer({ storage });

// ✅ Basic Upload Route
router.post("/", upload.single("file"), async (req, res) => {
  try {
    console.log("Received file:", req.file);
    console.log("Received body:", req.body);

    if (!req.file) {
      console.log("❌ No file uploaded");
      return res.status(400).json({ message: "No file uploaded" });
    }

    // ✅ File path for local storage
```

```
        // const fileUrl = http://localhost:5000/uploads/${req.file.filename};
        // const fileUrl = http://localhost:5000/uploads/${req.file.filename};
        const fileUrl = req.file.filename;
        // ✅ Check if metadata exists
        const { title, description, uploadedBy } = req.body;
        if (!title || !description) {
          console.log("❌ Missing title or description");
          return res.status(400).json({ message: "Title and description are required" });
        }

        // ✅ Ensure uploadedBy is a string (or default to "Unknown")
        const newResource = new Resource({
          title,
          description,
          fileUrl,
          uploadedBy: uploadedBy || "Unknown"
        });

        console.log("Saving to DB:", newResource); // ✅ Debugging

        await newResource.save(); // ✅ This must execute
        console.log("✅ Resource saved successfully!");

        res.status(201).json({
          success: true,
          message: "Resource uploaded and saved successfully!",
           fileUrl // ✅ Full URL
        });

    } catch (error) {
        console.error("❌ Upload Error:", error);
        res.status(500).json({ message: "Error uploading resource", error });
    }
});

// ✅ Fetch all resources
router.get("/", async (req, res) => {
    try {
        const resources = await Resource.find();
        console.log("Fetched resources:", resources); // 🔍 Debugging log
        res.json(resources);
    } catch (error) {
        console.error("❌ Error fetching resources:", error);
        res.status(500).json({ message: "Error fetching resources" });
    }
});

module.exports = router;
```