LoginForm

```
import React, { useState } from 'react';
import axios from 'axios';
import { Form, Input, Button, message, Alert } from 'antd';
import { UserOutlined, LockOutlined, SafetyOutlined } from '@ant-design/icons';
import "/src/Login.css";

const Login = ({ onLogin }) => {
  const [form] = Form.useForm();
  const [loading, setLoading] = useState(false);
  const [verificationMode, setVerificationMode] = useState(false);
  const [username, setUsername] = useState('');
  const [error, setError] = useState('');

  const handleLoginRequest = async (values) => {
    setLoading(true);
    setError('');

    try {
      console.log('Sending login request with:', values.username);
      const response = await axios.post('http://localhost:5000/api/login/request', values);
      console.log('Login request response:', response.data);

      setUsername(values.username);
      setVerificationMode(true);
      message.success('Verification code sent to your email');
```

```
    } catch (error) {
      console.error('Login error:', error);
      setError(error.response?.data?.message || 'Login failed. Please check your credentials and try again.');
      message.error(error.response?.data?.message || 'Login failed');
    } finally {
      setLoading(false);
    }
  };

  const [verificationCode, setVerificationCode] = useState('');

  const handleVerify = async () => {
    setLoading(true);
    setError('');

    try {
      console.log('Sending verification code:', verificationCode);
      const response = await axios.post("http://localhost:5000/api/login/verify", {
        username,
        code: verificationCode,
      });

      console.log('Verification response:', response.data);
      localStorage.setItem("authToken", response.data.token);
      localStorage.setItem("username", response.data.username);
```

```
    localStorage.setItem("role", response.data.role);

    message.success("Login successful");

    // Use onLogin prop if available, otherwise use window.location
    if (onLogin) {
      onLogin(response.data.token);
    } else {
      window.location.href = "/dashboard";
    }
  } catch (error) {
    console.error('Verification error:', error);
    setError(error.response?.data?.message || "Verification failed. Please try again.");
    message.error(error.response?.data?.message || "Verification failed. Please try again.");
  } finally {
    setLoading(false);
  }
};

return (
  <div className="login-container">
    {error && <Alert message={error} type="error" showIcon style={{ marginBottom: 16 }} />}

    {!verificationMode ? (
      <Form
        form={form}
```

```
  name="login"
  onFinish={handleLoginRequest}
  className="login-form"
>
  <Form.Item
    name="username"
    rules={[{ required: true, message: 'Please input your username!' }]}
  >
    <Input
      prefix={<UserOutlined className="site-form-item-icon" />}
      placeholder="Username"
      autoComplete="username"
    />
  </Form.Item>

  <Form.Item
    name="password"
    rules={[{ required: true, message: 'Please input your password!' }]}
  >
    <Input.Password
      prefix={<LockOutlined className="site-form-item-icon" />}
      placeholder="Password"
      autoComplete="current-password"
    />
  </Form.Item>
```

```
    <Form.Item>
      <Button
        type="primary"
        htmlType="submit"
        loading={loading}
        className="login-form-button"
        block
      >
        Continue
      </Button>
    </Form.Item>
  </Form>
) : (
  <div className="verification-form">
    <h3>Verification Required</h3>
    <p>Please enter the verification code sent to your email:</p>
    <Input
      prefix={<SafetyOutlined className="site-form-item-icon" />}
      placeholder="6-digit code"
      value={verificationCode}
      onChange={(e) => setVerificationCode(e.target.value)}
      style={{ marginBottom: 16 }}
      maxLength={6}
    />
    <Button
      type="primary"
```

```
          onClick={handleVerify}
          loading={loading}
          className="login-form-button"
          block
        >
          Verify & Login
        </Button>
      </div>
    )}
  </div>
 );
};
export default Login;
```

app.jsx

```
import React, { useState, useEffect } from "react";
import { BrowserRouter as Router, Routes, Route, Navigate } from "react-router-dom";
import { Layout, Button } from "antd";
import Sidebar from "./components/Sidebar";
import CustomHeader from "./components/Header";
import Dashboard from "./pages/Dashboard";
import Users from "./pages/Users";
import Schedule from "./pages/Schedule";
import Events from "./pages/Events";
import Resource from "./pages/Resource";
import Communication from "./pages/Communication";
import Reports from "./pages/Reports";
```

```
import Settings from "./pages/Settings";
import Logout from "./pages/Logout";
import Login from "./Login";
import { jwtDecode } from "jwt-decode";
import "./App.css";

const { Sider, Header, Content } = Layout;

const App = () => {
  const [isAuthenticated, setIsAuthenticated] = useState(false);
  const [loading, setLoading] = useState(true);

  useEffect(() => {
    checkAuth();
    setLoading(false);
  }, []);

  const checkAuth = () => {
    const token = localStorage.getItem("authToken");
    if (token) {
      try {
        const decoded = jwtDecode(token);
        if (decoded.exp > Date.now() / 1000) {
          setIsAuthenticated(true);
          return;
        }
```

```
    } catch { }
  }
  localStorage.removeItem("authToken");
  setIsAuthenticated(false);
};

const handleLogin = (token) => {
  localStorage.setItem("authToken", token);
  setIsAuthenticated(true);
  window.location.href = "/";
};

const handleLogout = () => {
  localStorage.removeItem("authToken");
  setIsAuthenticated(false);
  window.location.href = "/login";
};

const AuthenticatedLayout = () => (
  <Layout style={{ minHeight: "100vh" }}>
    <Sidebar />
    <Layout className="site-layout">
      <CustomHeader>
        <Button type="primary" onClick={handleLogout}>
          Logout
        </Button>
```

```
      </CustomHeader>
      <Content
        className="site-layout-background"
        style={{
          margin: "24px 16px",
          padding: 24,
          minHeight: 280,
        }}
      >
        <Routes>
          <Route path="/" element={<Dashboard />} />
          <Route path="/users" element={<Users />} />
          <Route path="/schedule" element={<Schedule />} />
          <Route path="/events" element={<Events />} />
          <Route path="/resource" element={<Resource />} />
          <Route path="/communication" element={<Communication />} />
          <Route path="/reports" element={<Reports />} />
          <Route path="/settings" element={<Settings />} />
          <Route path="/logout" element={<Logout onLogout={handleLogout} />} />
        </Routes>
      </Content>
    </Layout>
  </Layout>
);

return loading ? <div>Loading...</div> : (
```

```
    <Router>
      <Routes>
        <Route path="/login" element={!isAuthenticated ? <Login onLogin={handleLogin} /> : <Navigate to="/" />} />
        <Route path="*" element={isAuthenticated ? <AuthenticatedLayout /> : <Navigate to="/login" />} />
      </Routes>
    </Router>
  );
};

export default App;
```

serverside
email.js (email verification)

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

const transporter = nodemailer.createTransport({
  service: 'gmail',
  auth: {
    user: process.env.EMAIL_USER,
    pass: process.env.EMAIL_PASS
  }
});
```

```
const verificationCodes = {};

// Generate a random 6-digit verification code
const generateVerificationCode = () => {
  return crypto.randomInt(100000, 999999).toString();
};

// Send verification email
const sendVerificationEmail = async (email, username) => {
  const code = generateVerificationCode();

  // Store verification code with expiry (5 minutes)
  verificationCodes[username] = {
    code,
    expiry: Date.now() + 5 * 60 * 1000
  };

  const mailOptions = {
    from: process.env.EMAIL_USER,
    to: email,
    subject: 'Login Verification Code',
    text: `Your verification code is: ${code}. This code will expire in 5 minutes.`
  };

  try {
    await transporter.sendMail(mailOptions);
```

```
    console.log(`Verification code sent to ${email}`);
  } catch (error) {
    console.error('Error sending email:', error);
    throw new Error('Could not send verification email');
  }
};

// Verify the code entered by the user
const verifyCode = (username, code) => {
  const storedData = verificationCodes[username];

  if (!storedData) {
    return false;
  }

  // Check expiry time
  if (Date.now() > storedData.expiry) {
    delete verificationCodes[username];
    return false;
  }

  // Validate code
  if (storedData.code === code) {
    delete verificationCodes[username]; // Remove after successful verification
    return true;
  }
```

```
  return false;
};

module.exports = { sendVerificationEmail, verifyCode };
```

loginRoute.js

```
const express = require("express");
const router = express.Router();
const Login = require("../models/Login_details.js");
const { sendVerificationEmail, verifyCode } = require("../middleware/email.js");
const jwt = require("jsonwebtoken");
const bcrypt = require("bcrypt");

// Add logging to help debug the issue
router.post("/request", async (req, res) => {
  try {
    const { username, password } = req.body;
    console.log(`Login attempt for username: ${username}`);

    const user = await Login.findOne({ username });

    if (!user) {
      console.log(`User not found: ${username}`);
      return res.status(401).json({ message: "Invalid credentials" });
    }
```

```
    console.log(`User found, comparing passwords`);
    const passwordMatch = await bcrypt.compare(password, user.password);

    if (!passwordMatch) {
      console.log(`Password mismatch for user: ${username}`);
      return res.status(401).json({ message: "Invalid credentials" });
    }

    console.log(`Password matched, sending verification email to: ${user.email}`);
    await sendVerificationEmail(user.email, username);
    res.status(200).json({ message: "Verification code sent" });
  } catch (error) {
    console.error("Login error:", error);
    res.status(500).json({ message: "Server error" });
  }
});

router.post("/verify", async (req, res) => {
  try {
    const { username, code } = req.body;
    console.log(`Verification attempt for: ${username} with code: ${code}`);

    if (!verifyCode(username, code)) {
      console.log(`Invalid or expired code for user: ${username}`);
      return res.status(401).json({ message: "Invalid or expired code" });
```

```
    }

    const user = await Login.findOne({ username });
    console.log(`User verified: ${username}, role: ${user.role}`);

    if (!process.env.JWT_SECRET) {
      console.error("JWT_SECRET environment variable not set");
      throw new Error("JWT_SECRET is not defined");
    }

    const token = jwt.sign(
      { username: user.username, role: user.role },
      process.env.JWT_SECRET,
      { expiresIn: "8h" }
    );

    res.status(200).json({ token, username: user.username, role: user.role });
  } catch (error) {
    console.error("Verification error:", error);
    res.status(500).json({ message: "Server error" });
  }
});

module.exports = router;
```

the index.js

```
require('dotenv').config();

const express = require('express');

const connectDB = require('./db');

const Login = require('./models/Login_details.js');

const EventModel = require('./models/Event.js');

const UserModel = require('./models/Users.js');

const ResourceModel = require('./models/resource.js');

const cors = require('cors');


const app = express();

app.use(express.json());

const corsOptions = {

   origin: ['http://localhost:5173', 'http://localhost:5174', 'http://localhost:5175',
'http://localhost:5176', 'http://localhost:5177', 'http://localhost:5178', 'http://localhost:5179',
'http://localhost:5180', 'http://localhost:5181', 'http://localhost:5182', 'http://localhost:5183',
'http://localhost:5184', 'http://localhost:5185'],

   methods: 'GET,POST,PUT,DELETE',

   allowedHeaders: 'Content-Type,Authorization',

   credentials: true

};


app.use(cors(corsOptions));

connectDB();


app.use("/api/resources", require("./routes/resourceRoute.js"));

app.use("/api/login", require("./routes/loginRoute.js"));
```

```
app.use("/api/users", require("./routes/userRoutes.js"));
app.use("/api/events", require("./routes/eventRoutes.js"));

app.get('/', (req, res) => {
   res.send('API is running...');
});

const PORT = process.env.PORT || 5000;

app.listen(PORT, () => console.log(`Server running on port ${PORT}`));
```