Admin Dashboard

To restrict access to the dashboard so only admins can access it, you can implement role-based authentication in your React application and backend. Here's how you can achieve this:

---

1. Backend: Define User Roles

Make sure your backend supports user roles. For example, a user object in your database could look like this:

```
{
  "id": 1,
  "username": "admin",
  "email": "admin@example.com",
  "role": "admin"
}
```

When authenticating a user, include their role in the token payload. For example, if you use JWT:

```
const jwt = require('jsonwebtoken');

// Generate token
const token = jwt.sign(
   { id: user.id, role: user.role }, // Include user role
   process.env.JWT_SECRET,
   { expiresIn: '1h' }
);
```

---

2. Frontend: Check User Role

(a) Protect the Dashboard Route

Use React Router's Navigate component to restrict access to the dashboard. For example:

```
import React from 'react';
import { Navigate } from 'react-router-dom';

const ProtectedRoute = ({ user, children }) => {
   if (!user) {
      return <Navigate to="/login" />;
   }

   if (user.role !== 'admin') {
      return <Navigate to="/not-authorized" />;
   }

   return children;
};

export default ProtectedRoute;
```

(b) Use ProtectedRoute for the Dashboard
Wrap your dashboard route with ProtectedRoute in your routing configuration:

```
import React from 'react';
import { BrowserRouter as Router, Routes, Route } from 'react-router-dom';
import Dashboard from './pages/Dashboard';
import Login from './pages/Login';
import NotAuthorized from './pages/NotAuthorized';
import Home from './pages/Home';
import ProtectedRoute from './components/ProtectedRoute';

const App = () => {
  const user = JSON.parse(localStorage.getItem('user')); // Get user info from local
storage or context

  return (
    <Router>
      <Routes>
        <Route path="/" element={<Home />} />
        <Route path="/login" element={<Login />} />
        <Route
          path="/dashboard"
          element={
            <ProtectedRoute user={user}>
              <Dashboard />
            </ProtectedRoute>
          }
        />
        <Route path="/not-authorized" element={<NotAuthorized />} />
      </Routes>
    </Router>
  );
};

export default App;
```

---

3. Backend: Protect the Dashboard API
In your backend, create middleware to verify the user's role before granting access to admin-only routes:

```
const jwt = require('jsonwebtoken');

const verifyAdmin = (req, res, next) => {
  const token = req.headers.authorization?.split(' ')[1];

  if (!token) {
    return res.status(401).json({ message: 'Unauthorized' });
  }
```

```
  try {
    const decoded = jwt.verify(token, process.env.JWT_SECRET);
    if (decoded.role !== 'admin') {
      return res.status(403).json({ message: 'Forbidden: Admins only' });
    }
    req.user = decoded;
    next();
  } catch (err) {
    res.status(403).json({ message: 'Forbidden' });
  }
};

module.exports = verifyAdmin;
```

Apply this middleware to your dashboard-related routes:

```
const express = require('express');
const router = express.Router();
const verifyAdmin = require('./middlewares/verifyAdmin');

router.get('/dashboard', verifyAdmin, (req, res) => {
  res.status(200).json({ message: 'Welcome, Admin!' });
});

module.exports = router;
```

---

4. User Login: Store Role in State

When a user logs in, save their role in local storage or global state management (e.g., Context API, Redux):

```
const handleLogin = async () => {
  const response = await fetch('/api/login', { /* login data */ });
  const data = await response.json();

  if (data.token) {
    localStorage.setItem('user', JSON.stringify({ token: data.token, role: data.role }));
    // Redirect to dashboard or other page
  }
};
```

---

5. Create a "Not Authorized" Page

Add a simple NotAuthorized page to inform non-admin users they cannot access the dashboard:

```
import React from 'react';

const NotAuthorized = () => {
  return (
    <div>
      <h1>403 - Not Authorized</h1>
      <p>You do not have permission to view this page.</p>
```

```
      </div>
    );
};

export default NotAuthorized;
```

Summary of Flow

1. **User Logs In**: Backend includes the user's role in the JWT or response.
2. **Role-Based Access**: Frontend verifies the role before rendering the dashboard.
3. **Route Protection**: Use ProtectedRoute for admin-only routes.
4. **API Protection**: Backend middleware ensures only admins can access sensitive endpoints.

Let me know if you need help with any specific part!