Lecture # 16

JSON Web Token (JWT)

Table of Contents

Overview:

JWT (JSON Web Token) authentication is a stateless authentication method that's perfect for modern web applications. This lessons covers:

- User registration and login
- JWT token generation and validation
- Protected routes on both backend and frontend
- Token refresh mechanism
- Logout functionality

Backend Setup (Node.js + Express):

---

1. Initialize Backend Project

```
mkdir mern-jwt-auth
cd mern-jwt-auth
mkdir backend
cd backend
npm init -y
```

---

2. Install Dependencies

```
npm install express mongoose bcryptjs jsonwebtoken cors dotenv
npm install -D nodemon
```

---

3. Project Structure

```
backend/
├── controllers/
│   └── authController.js
├── middleware/
│   └── authMiddleware.js
├── models/
│   └── User.js
├── routes/
│   └── auth.js
├── .env
├── server.js
└── package.json
```

4. Environment Variables (.env)

```
PORT=5000
MONGODB_URI=mongodb://localhost:27017/mern-jwt-auth
JWT_SECRET=your-super-secret-jwt-key-here-make-it-long-and-complex
JWT_EXPIRE=7d
JWT_REFRESH_SECRET=your-refresh-token-secret-here
JWT_REFRESH_EXPIRE=30d
NODE_ENV=development
```

5. Server Setup (server.js)

```
const express = require('express');
const mongoose = require('mongoose');
const cors = require('cors');
const dotenv = require('dotenv');

// Load environment variables
dotenv.config();

const app = express();

// Middleware
app.use(cors({
  origin: 'http://localhost:3000', // React app URL
  credentials: true
}));
app.use(express.json());

// Routes
app.use('/api/auth', require('./routes/auth'));

// Error handling middleware
app.use((err, req, res, next) => {
```

```
  console.error(err.stack);
  res.status(500).json({ message: 'Something went wrong!' });
});

// Connect to MongoDB
mongoose.connect(process.env.MONGODB_URI)
  .then(() => console.log('MongoDB connected'))
  .catch(err => console.log('MongoDB connection error:', err));

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

---

6. User Model (models/User.js)

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

const userSchema = new mongoose.Schema({
  name: {
    type: String,
    required: [true, 'Please provide a name'],
    trim: true
  },
  email: {
    type: String,
    required: [true, 'Please provide an email'],
    unique: true,
    lowercase: true,
    match: [
      /^\w+([.-]?\w+)*@\w+([.-]?\w+)*(\.\w{2,3})+$/,
      'Please provide a valid email'
    ]
  },
  password: {
    type: String,
    required: [true, 'Please provide a password'],
    minlength: 6,
    select: false
  },
  refreshToken: {
    type: String,
    select: false
  }
}, {
  timestamps: true
});

// Hash password before saving
userSchema.pre('save', async function(next) {
  if (!this.isModified('password')) return next();
```

```
  this.password = await bcrypt.hash(this.password, 12);
  next();
});

// Compare password method
userSchema.methods.comparePassword = async function(candidatePassword) {
  return await bcrypt.compare(candidatePassword, this.password);
};

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

---

7. Authentication Middleware (middleware/authMiddleware.js)

```
const jwt = require('jsonwebtoken');
const User = require('../models/User');

const protect = async (req, res, next) => {
  try {
    let token;

    // Check if token exists in headers
    if (req.headers.authorization && req.headers.authorization.startsWith('Bearer')) {
      token = req.headers.authorization.split(' ')[1];
    }

    if (!token) {
      return res.status(401).json({ message: 'Not authorized, no token' });
    }

    // Verify token
    const decoded = jwt.verify(token, process.env.JWT_SECRET);

    // Get user from token
    req.user = await User.findById(decoded.id).select('-password');

    if (!req.user) {
      return res.status(401).json({ message: 'User not found' });
    }

    next();
  } catch (error) {
    console.error('Auth middleware error:', error);

    if (error.name === 'JsonWebTokenError') {
      return res.status(401).json({ message: 'Not authorized, invalid token' });
    } else if (error.name === 'TokenExpiredError') {
      return res.status(401).json({ message: 'Not authorized, token expired' });
    }

    res.status(401).json({ message: 'Not authorized' });
  }
```

```
};

module.exports = { protect };
```

8. Authentication Controller (controllers/authController.js)

```
const jwt = require('jsonwebtoken');
const User = require('../models/User');

// Generate JWT Token
const generateToken = (id) => {
  return jwt.sign({ id }, process.env.JWT_SECRET, {
    expiresIn: process.env.JWT_EXPIRE
  });
};

// Generate Refresh Token
const generateRefreshToken = (id) => {
  return jwt.sign({ id }, process.env.JWT_REFRESH_SECRET, {
    expiresIn: process.env.JWT_REFRESH_EXPIRE
  });
};

// Register User
const register = async (req, res) => {
  try {
    const { name, email, password } = req.body;

    // Validation
    if (!name || !email || !password) {
      return res.status(400).json({ message: 'Please provide all required fields' });
    }

    if (password.length < 6) {
      return res.status(400).json({ message: 'Password must be at least 6 characters'
});
    }

    // Check if user exists
    const existingUser = await User.findOne({ email });
    if (existingUser) {
      return res.status(400).json({ message: 'User already exists' });
    }

    // Create user
    const user = await User.create({ name, email, password });

    // Generate tokens
    const token = generateToken(user._id);
    const refreshToken = generateRefreshToken(user._id);

    // Save refresh token to user
```

```
    user.refreshToken = refreshToken;
    await user.save();

    res.status(201).json({
      success: true,
      token,
      refreshToken,
      user: {
        id: user._id,
        name: user.name,
        email: user.email
      }
    });
  } catch (error) {
    console.error('Register error:', error);
    res.status(500).json({ message: 'Server error' });
  }
};

// Login User
const login = async (req, res) => {
  try {
    const { email, password } = req.body;

    // Validation
    if (!email || !password) {
      return res.status(400).json({ message: 'Please provide email and password' });
    }

    // Check user exists and get password
    const user = await User.findOne({ email }).select('+password');
    if (!user) {
      return res.status(401).json({ message: 'Invalid credentials' });
    }

    // Check password
    const isPasswordCorrect = await user.comparePassword(password);
    if (!isPasswordCorrect) {
      return res.status(401).json({ message: 'Invalid credentials' });
    }

    // Generate tokens
    const token = generateToken(user._id);
    const refreshToken = generateRefreshToken(user._id);

    // Save refresh token to user
    user.refreshToken = refreshToken;
    await user.save();

    res.json({
      success: true,
      token,
      refreshToken,
      user: {
```

```
        id: user._id,
        name: user.name,
        email: user.email
      }
    });
  } catch (error) {
    console.error('Login error:', error);
    res.status(500).json({ message: 'Server error' });
  }
};

// Get Current User
const getMe = async (req, res) => {
  try {
    const user = await User.findById(req.user.id);
    res.json({
      success: true,
      user: {
        id: user._id,
        name: user.name,
        email: user.email
      }
    });
  } catch (error) {
    console.error('Get me error:', error);
    res.status(500).json({ message: 'Server error' });
  }
};

// Refresh Token
const refreshToken = async (req, res) => {
  try {
    const { refreshToken } = req.body;

    if (!refreshToken) {
      return res.status(401).json({ message: 'Refresh token not provided' });
    }

    // Verify refresh token
    const decoded = jwt.verify(refreshToken, process.env.JWT_REFRESH_SECRET);

    // Find user with this refresh token
    const user = await User.findOne({
      _id: decoded.id,
      refreshToken: refreshToken
    });

    if (!user) {
      return res.status(401).json({ message: 'Invalid refresh token' });
    }

    // Generate new tokens
    const newToken = generateToken(user._id);
    const newRefreshToken = generateRefreshToken(user._id);
```

```
    // Save new refresh token
    user.refreshToken = newRefreshToken;
    await user.save();

    res.json({
      success: true,
      token: newToken,
      refreshToken: newRefreshToken
    });
  } catch (error) {
    console.error('Refresh token error:', error);

    if (error.name === 'JsonWebTokenError' || error.name === 'TokenExpiredError') {
      return res.status(401).json({ message: 'Invalid refresh token' });
    }

    res.status(500).json({ message: 'Server error' });
  }
};

// Logout
const logout = async (req, res) => {
  try {
    // Clear refresh token from database
    await User.findByIdAndUpdate(req.user.id, { refreshToken: null });

    res.json({
      success: true,
      message: 'Logged out successfully'
    });
  } catch (error) {
    console.error('Logout error:', error);
    res.status(500).json({ message: 'Server error' });
  }
};

module.exports = {
  register,
  login,
  getMe,
  refreshToken,
  logout
};
```

---

9. Authentication Routes (routes/auth.js)

```
const express = require('express');
const {
  register,
  login,
  getMe,
```

```
  refreshToken,
  logout
} = require('../controllers/authController');
const { protect } = require('../middleware/authMiddleware');

const router = express.Router();

router.post('/register', register);
router.post('/login', login);
router.post('/refresh-token', refreshToken);
router.get('/me', protect, getMe);
router.post('/logout', protect, logout);

module.exports = router;
```

---

10. Update package.json Scripts

```
{
  "scripts": {
    "start": "node server.js",
    "dev": "nodemon server.js"
  }
}
```

Frontend Setup (React)

---

1. Initialize React App

```
cd ..
npx create-react-app frontend
cd frontend
npm install axios react-router-dom
```

---

2. Project Structure

```
frontend/src/
├── components/
│   ├── Login.js
│   ├── Register.js
│   ├── Dashboard.js
│   └── PrivateRoute.js
├── context/
│   └── AuthContext.js
├── services/
│   └── api.js
├── App.js
```

```
    └── index.js
```

---

3. API Service (services/api.js)

```
import axios from 'axios';

const API_URL = 'http://localhost:5000/api';

// Create axios instance
const api = axios.create({
  baseURL: API_URL,
  headers: {
    'Content-Type': 'application/json',
  },
});

// Request interceptor to add token
api.interceptors.request.use(
  (config) => {
    const token = localStorage.getItem('token');
    if (token) {
      config.headers.Authorization = `Bearer ${token}`;
    }
    return config;
  },
  (error) => {
    return Promise.reject(error);
  }
);

// Response interceptor for token refresh
api.interceptors.response.use(
  (response) => response,
  async (error) => {
    const originalRequest = error.config;

    if (error.response?.status === 401 && !originalRequest._retry) {
      originalRequest._retry = true;

      try {
        const refreshToken = localStorage.getItem('refreshToken');
        if (refreshToken) {
          const response = await axios.post(`${API_URL}/auth/refresh-token`, {
            refreshToken,
          });

          const { token, refreshToken: newRefreshToken } = response.data;
          localStorage.setItem('token', token);
          localStorage.setItem('refreshToken', newRefreshToken);

          // Retry original request with new token
          originalRequest.headers.Authorization = `Bearer ${token}`;
```

```
          return api(originalRequest);
        }
      } catch (refreshError) {
        // Refresh failed, redirect to login
        localStorage.removeItem('token');
        localStorage.removeItem('refreshToken');
        window.location.href = '/login';
        return Promise.reject(refreshError);
      }
    }

    return Promise.reject(error);
  }
);

export const authAPI = {
  register: (userData) => api.post('/auth/register', userData),
  login: (userData) => api.post('/auth/login', userData),
  getMe: () => api.get('/auth/me'),
  logout: () => api.post('/auth/logout'),
  refreshToken: (refreshToken) => api.post('/auth/refresh-token', { refreshToken }),
};

export default api;
```

---

4. Auth Context (context/AuthContext.js)

```
import React, { createContext, useContext, useReducer, useEffect } from 'react';
import { authAPI } from '../services/api';

const AuthContext = createContext();

const initialState = {
  user: null,
  token: localStorage.getItem('token'),
  refreshToken: localStorage.getItem('refreshToken'),
  loading: true,
  error: null,
};

const authReducer = (state, action) => {
  switch (action.type) {
    case 'SET_LOADING':
      return { ...state, loading: action.payload };
    case 'SET_ERROR':
      return { ...state, error: action.payload, loading: false };
    case 'LOGIN_SUCCESS':
    case 'REGISTER_SUCCESS':
      localStorage.setItem('token', action.payload.token);
      localStorage.setItem('refreshToken', action.payload.refreshToken);
      return {
        ...state,
```

```
        user: action.payload.user,
        token: action.payload.token,
        refreshToken: action.payload.refreshToken,
        loading: false,
        error: null,
      };
    case 'LOAD_USER':
      return {
        ...state,
        user: action.payload,
        loading: false,
      };
    case 'LOGOUT':
      localStorage.removeItem('token');
      localStorage.removeItem('refreshToken');
      return {
        ...state,
        user: null,
        token: null,
        refreshToken: null,
        loading: false,
      };
    case 'CLEAR_ERROR':
      return { ...state, error: null };
    default:
      return state;
  }
};

export const AuthProvider = ({ children }) => {
  const [state, dispatch] = useReducer(authReducer, initialState);

  // Load user on app start
  useEffect(() => {
    const loadUser = async () => {
      if (state.token) {
        try {
          const response = await authAPI.getMe();
          dispatch({ type: 'LOAD_USER', payload: response.data.user });
        } catch (error) {
          console.error('Load user error:', error);
          dispatch({ type: 'LOGOUT' });
        }
      } else {
        dispatch({ type: 'SET_LOADING', payload: false });
      }
    };

    loadUser();
  }, [state.token]);

  const register = async (userData) => {
    try {
      dispatch({ type: 'SET_LOADING', payload: true });
```

```
      const response = await authAPI.register(userData);
      dispatch({ type: 'REGISTER_SUCCESS', payload: response.data });
      return { success: true };
    } catch (error) {
      const message = error.response?.data?.message || 'Registration failed';
      dispatch({ type: 'SET_ERROR', payload: message });
      return { success: false, error: message };
    }
  };

  const login = async (userData) => {
    try {
      dispatch({ type: 'SET_LOADING', payload: true });
      const response = await authAPI.login(userData);
      dispatch({ type: 'LOGIN_SUCCESS', payload: response.data });
      return { success: true };
    } catch (error) {
      const message = error.response?.data?.message || 'Login failed';
      dispatch({ type: 'SET_ERROR', payload: message });
      return { success: false, error: message };
    }
  };

  const logout = async () => {
    try {
      await authAPI.logout();
    } catch (error) {
      console.error('Logout error:', error);
    } finally {
      dispatch({ type: 'LOGOUT' });
    }
  };

  const clearError = () => {
    dispatch({ type: 'CLEAR_ERROR' });
  };

  const value = {
    user: state.user,
    token: state.token,
    loading: state.loading,
    error: state.error,
    register,
    login,
    logout,
    clearError,
    isAuthenticated: !!state.token && !!state.user,
  };

  return <AuthContext.Provider value={value}>{children}</AuthContext.Provider>;
};

export const useAuth = () => {
  const context = useContext(AuthContext);
```

```
  if (!context) {
    throw new Error('useAuth must be used within an AuthProvider');
  }
  return context;
};
```

5. Private Route Component (components/PrivateRoute.js)

```
import React from 'react';
import { Navigate, useLocation } from 'react-router-dom';
import { useAuth } from '../context/AuthContext';

const PrivateRoute = ({ children }) => {
  const { isAuthenticated, loading } = useAuth();
  const location = useLocation();

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

  return isAuthenticated ? (
    children
  ) : (
    <Navigate to="/login" state={{ from: location }} replace />
  );
};

export default PrivateRoute;
```

6. Register Component (components/Register.js)

```
import React, { useState, useEffect } from 'react';
import { Link, useNavigate } from 'react-router-dom';
import { useAuth } from '../context/AuthContext';

const Register = () => {
  const [formData, setFormData] = useState({
    name: '',
    email: '',
    password: '',
    confirmPassword: '',
  });
  const [localError, setLocalError] = useState('');

  const { register, loading, error, clearError, isAuthenticated } = useAuth();
  const navigate = useNavigate();

  useEffect(() => {
    if (isAuthenticated) {
```

```
    navigate('/dashboard');
  }
}, [isAuthenticated, navigate]);

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

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

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

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

  const result = await register({
    name: formData.name,
    email: formData.email,
    password: formData.password,
  });

  if (result.success) {
    navigate('/dashboard');
  }
};

return (
  <div className="auth-container">
    <div className="auth-form">
      <h2>Register</h2>
      {(error || localError) && (
        <div className="error-message">{localError || error}</div>
      )}
      <form onSubmit={handleSubmit}>
        <div className="form-group">
          <label htmlFor="name">Name</label>
          <input
            type="text"
            id="name"
            name="name"
            value={formData.name}
            onChange={handleChange}
            required
          />
        </div>
        <div className="form-group">
          <label htmlFor="email">Email</label>
```

```
            <input
              type="email"
              id="email"
              name="email"
              value={formData.email}
              onChange={handleChange}
              required
            />
          </div>
          <div className="form-group">
            <label htmlFor="password">Password</label>
            <input
              type="password"
              id="password"
              name="password"
              value={formData.password}
              onChange={handleChange}
              required
              minLength="6"
            />
          </div>
          <div className="form-group">
            <label htmlFor="confirmPassword">Confirm Password</label>
            <input
              type="password"
              id="confirmPassword"
              name="confirmPassword"
              value={formData.confirmPassword}
              onChange={handleChange}
              required
            />
          </div>
          <button type="submit" disabled={loading} className="auth-button">
            {loading ? 'Registering...' : 'Register'}
          </button>
        </form>
        <p>
          Already have an account? <Link to="/login">Login here</Link>
        </p>
      </div>
    </div>
  );
};

export default Register;
```

---

7. Login Component (components/Login.js)

```
import React, { useState, useEffect } from 'react';
import { Link, useNavigate, useLocation } from 'react-router-dom';
import { useAuth } from '../context/AuthContext';
```

```
const Login = () => {
  const [formData, setFormData] = useState({
    email: '',
    password: '',
  });

  const { login, loading, error, clearError, isAuthenticated } = useAuth();
  const navigate = useNavigate();
  const location = useLocation();

  const from = location.state?.from?.pathname || '/dashboard';

  useEffect(() => {
    if (isAuthenticated) {
      navigate(from, { replace: true });
    }
  }, [isAuthenticated, navigate, from]);

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

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

  const handleSubmit = async (e) => {
    e.preventDefault();
    const result = await login(formData);
    if (result.success) {
      navigate(from, { replace: true });
    }
  };

  return (
    <div className="auth-container">
      <div className="auth-form">
        <h2>Login</h2>
        {error && <div className="error-message">{error}</div>}
        <form onSubmit={handleSubmit}>
          <div className="form-group">
            <label htmlFor="email">Email</label>
            <input
              type="email"
              id="email"
              name="email"
              value={formData.email}
              onChange={handleChange}
              required
            />
          </div>
          <div className="form-group">
            <label htmlFor="password">Password</label>
```

```
            <input
              type="password"
              id="password"
              name="password"
              value={formData.password}
              onChange={handleChange}
              required
            />
          </div>
          <button type="submit" disabled={loading} className="auth-button">
            {loading ? 'Logging in...' : 'Login'}
          </button>
        </form>
        <p>
          Don't have an account? <Link to="/register">Register here</Link>
        </p>
      </div>
    </div>
  );
};

export default Login;
```

---

8. Dashboard Component (components/Dashboard.js)

```
import React from 'react';
import { useAuth } from '../context/AuthContext';

const Dashboard = () => {
  const { user, logout } = useAuth();

  const handleLogout = () => {
    logout();
  };

  return (
    <div className="dashboard">
      <div className="dashboard-header">
        <h1>Dashboard</h1>
        <button onClick={handleLogout} className="logout-button">
          Logout
        </button>
      </div>
      <div className="dashboard-content">
        <div className="user-info">
          <h2>Welcome, {user?.name}!</h2>
          <p>Email: {user?.email}</p>
          <p>User ID: {user?.id}</p>
        </div>
        <div className="dashboard-features">
          <h3>Protected Content</h3>
          <p>This content is only visible to authenticated users.</p>
```

```
          <div className="feature-cards">
            <div className="feature-card">
              <h4>Profile Management</h4>
              <p>Update your profile information</p>
            </div>
            <div className="feature-card">
              <h4>Settings</h4>
              <p>Manage your account settings</p>
            </div>
            <div className="feature-card">
              <h4>Analytics</h4>
              <p>View your usage analytics</p>
            </div>
          </div>
        </div>
      </div>
    </div>
  );
};

export default Dashboard;
```

---

9. Main App Component (App.js)

```
import React from 'react';
import { BrowserRouter as Router, Routes, Route, Navigate } from 'react-router-dom';
import { AuthProvider } from './context/AuthContext';
import PrivateRoute from './components/PrivateRoute';
import Login from './components/Login';
import Register from './components/Register';
import Dashboard from './components/Dashboard';
import './App.css';

function App() {
  return (
    <AuthProvider>
      <Router>
        <div className="App">
          <Routes>
            <Route path="/login" element={<Login />} />
            <Route path="/register" element={<Register />} />
            <Route
              path="/dashboard"
              element={
                <PrivateRoute>
                  <Dashboard />
                </PrivateRoute>
              }
            />
            <Route path="/" element={<Navigate to="/dashboard" replace />} />
          </Routes>
        </div>
```

```
      </Router>
    </AuthProvider>
  );
}

export default App;
```

---

10. Styling (App.css)

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

body {
  font-family: -apple-system, BlinkMacSystemFont, 'Segoe UI', 'Roboto', 'Oxygen',
    'Ubuntu', 'Cantarell', 'Fira Sans', 'Droid Sans', 'Helvetica Neue',
    sans-serif;
  -webkit-font-smoothing: antialiased;
  -moz-osx-font-smoothing: grayscale;
  background-color: #f5f5f5;
}

.App {
  min-height: 100vh;
}

/* Auth Styles */
.auth-container {
  display: flex;
  justify-content: center;
  align-items: center;
  min-height: 100vh;
  padding: 20px;
}

.auth-form {
  background: white;
  padding: 2rem;
  border-radius: 8px;
  box-shadow: 0 2px 10px rgba(0, 0, 0, 0.1);
  width: 100%;
  max-width: 400px;
}

.auth-form h2 {
  text-align: center;
  margin-bottom: 1.5rem;
  color: #333;
}
```

```
.form-group {
  margin-bottom: 1rem;
}

.form-group label {
  display: block;
  margin-bottom: 0.5rem;
  font-weight: 500;
  color: #555;
}

.form-group input {
  width: 100%;
  padding: 0.75rem;
  border: 1px solid #ddd;
  border-radius: 4px;
  font-size: 1rem;
  transition: border-color 0.3s;
}

.form-group input:focus {
  outline: none;
  border-color: #007bff;
}

.auth-button {
  width: 100%;
  padding: 0.75rem;
  background-color: #007bff;
  color: white;
  border: none;
  border-radius: 4px;
  font-size: 1rem;
  cursor: pointer;
  transition: background-color 0.3s;
}

.auth-button:hover:not(:disabled) {
  background-color: #0056b3;
}

.auth-button:disabled {
  background-color: #6c757d;
  cursor: not-allowed;
}

.auth-form p {
  text-align: center;
  margin-top: 1rem;
  color: #666;
}

.auth-form a {
  color: #007bff;
```

```
  text-decoration: none;
}

.auth-form a:hover {
  text-decoration: underline;
}

.error-message {
  background-color: #f8d7da;
  color: #721c24;
  padding: 0.75rem;
  border: 1px solid #f5c6cb;
  border-radius: 4px;
  margin-bottom: 1rem;
  text-align: center;
}

/* Dashboard Styles */
.dashboard {
  min-height: 100vh;
  background-color: #f8f9fa;
}

.dashboard-header {
  background-color: white;
  padding: 1rem 2rem;
  border-bottom: 1px solid #dee2e6;
  display: flex;
  justify-content: space-between;
  align-items: center;
  box-shadow: 0 2px 4px rgba(0, 0, 0, 0.1);
}

.dashboard-header h1 {
  color: #333;
  margin: 0;
}

.logout-button {
  padding: 0.5rem 1rem;
  background-color: #dc3545;
  color: white;
  border: none;
  border-radius: 4px;
  cursor: pointer;
  transition: background-color 0.3s;
}

.logout-button:hover {
  background-color: #c82333;
}

.dashboard-content {
  padding: 2rem;
```

```
  max-width: 1200px;
  margin: 0 auto;
}

.user-info {
  background: white;
  padding: 2rem;
  border-radius: 8px;
  box-shadow: 0 2px 4px rgba(0, 0, 0, 0.1);
  margin-bottom: 2rem;
}

.user-info h2 {
  color: #333;
  margin-bottom: 1rem;
}

.user-info p {
  margin-bottom: 0.5rem;
  color: #666;
}

.dashboard-features {
  background: white;
  padding: 2rem;
  border-radius: 8px;
  box-shadow: 0 2px 4px rgba(0, 0, 0, 0.1);
}

.dashboard-features h3 {
  color: #333;
  margin-bottom: 1rem;
}

.dashboard-features > p {
  color: #666;
  margin-bottom: 2rem;
}

.feature-cards {
  display: grid;
  grid-template-columns: repeat(auto-fit, minmax(250px, 1fr));
  gap: 1rem;
}

.feature-card {
  background: #f8f9fa;
  padding: 1.5rem;
  border-radius: 6px;
  border: 1px solid #e9ecef;
}

.feature-card h4 {
  color: #333;
```

```
  margin-bottom: 0.5rem;
}

.feature-card p {
  color: #666;
  margin: 0;
}

/* Loading Styles */
.loading {
  display: flex;
  justify-content: center;
  align-items: center;
  min-height: 100vh;
  font-size: 1.2rem;
  color: #666;
}

/* Responsive Design */
@media (max-width: 768px) {
  .dashboard-header {
    padding: 1rem;
    flex-direction: column;
    gap: 1rem;
  }

  .dashboard-content {
    padding: 1rem;
  }

  .feature-cards {
    grid-template-columns: 1fr;
  }

  .auth-container {
    padding: 1rem;
  }
}
```

Testing the Application

1. Start the Backend Server

```
cd backend
npm run dev
```

2. Start the React App

```
cd frontend
npm start
```

---

3. Test the Authentication Flow

---

1. **Register a new user**:
   - Navigate to http://localhost:3000/register
   - Fill in the registration form
   - Verify redirection to dashboard
2. **Login with existing user**:
   - Navigate to http://localhost:3000/login
   - Enter credentials
   - Verify successful login and dashboard access
3. **Test protected routes**:
   - Try accessing /dashboard without authentication
   - Verify redirection to login page
4. **Test token refresh**:
   - Wait for token expiration or manually expire token
   - Make a request to protected route
   - Verify automatic token refresh
5. **Test logout**:
   - Click logout button
   - Verify redirection to login page
   - Try accessing protected routes (should be redirected)

---

4. API Testing with Postman

```
# Register
POST http://localhost:5000/api/auth/register
Content-Type: application/json

{
  "name": "John Doe",
  "email": "john@example.com",
  "password": "password123"
}

# Login
POST http://localhost:5000/api/auth/login
Content-Type: application/json

{
  "email": "john@example.com",
  "password": "password123"
}

# Get Current User (Protected)
```

```
GET http://localhost:5000/api/auth/me
Authorization: Bearer YOUR_JWT_TOKEN

# Refresh Token
POST http://localhost:5000/api/auth/refresh-token
Content-Type: application/json

{
  "refreshToken": "YOUR_REFRESH_TOKEN"
}

# Logout (Protected)
POST http://localhost:5000/api/auth/logout
Authorization: Bearer YOUR_JWT_TOKEN
```

Security Best Practices

1. Password Security

- Hash passwords with bcrypt (salt rounds: 12+)
- Implement password strength requirements
- Consider implementing password reset functionality

2. JWT Security

- Keep token expiration times short (15-30 minutes for access tokens)
- Use longer expiration for refresh tokens (7-30 days)
- Implement token blacklisting for logout
- Store tokens securely on the client side

3. CORS Configuration

```
app.use(cors({
  origin: process.env.CLIENT_URL || 'http://localhost:3000',
  credentials: true,
  optionsSuccessStatus: 200
}));
```