Google SSO Authentication Flow with Firebase and JWT

Overview

This document outlines the authentication flow for signing in users via **Google Single Sign-On (SSO)** using **Firebase Authentication** and **JWT (JSON Web Token)** for session management.

Flow Steps

1. **User clicks "Sign in with Google"**
2. **Firebase triggers Google sign-in popup**
3. **User selects a Google account and grants permissions**
4. **Firebase returns an ID token**
5. **Frontend sends the ID token to the backend API**
6. **Backend verifies the ID token using Firebase Admin SDK**
7. **User data is retrieved or created in the database**
8. **Backend generates a JWT token for session management**
9. **JWT token is stored in HTTP-only cookies**
10. **User successfully logs in and receives profile data**

Backend Implementation

The backend validates the Firebase ID token and issues a JWT token.

API: Google Login Endpoint

Route: `/auth/google`

Method: `POST`

Request Body:

```
{
  "idToken": "<Firebase_ID_Token>"
}
```

Response:

```
{
  "authenticated": true,
  "id": "<user_id>",
```

```
  "email": "user@example.com",
  "name": "User Name",
  "preferences": {},
  "message": "Login successful."
}
```

Code Implementation:

```
export const googleLogin = async (req, res) => {
  try {
    const { idToken } = req.body;

    if (!idToken) {
      return res.status(400).json({ message: 'ID token is required' });
    }

    // Verify the Firebase ID token
    const decodedToken = await admin.auth().verifyIdToken(idToken);
    console.log(decodedToken);

    // Check if the user exists in the database
    let user = await User.findOne({ email: decodedToken.email });

    if (!user) {
      user = new User({
        name: decodedToken.name || 'No Name',
        email: decodedToken.email,
        password: 'google-auth',
      });
      await user.save();
    }

    // Generate JWT Token
    const token = jwt.sign(
      { id: user._id, name: user.name, email: user.email },
      process.env.JWT_SECRET || 'hello_this_string',
      { expiresIn: '1d' }
    );

    // Set cookie with JWT Token
    res.cookie('token', token, {
      httpOnly: true,
      secure: process.env.NODE_ENV === 'production',
      maxAge: 15 * 24 * 60 * 60 * 1000,
    });

    res.status(200).json({
```

```
      authenticated: true,
      id: user._id,
      email: user.email,
      name: user.name,
      preferences: user.preferences || {},
      message: 'Login successful.',
    });
  } catch (err) {
    console.error('Google Login Error:', err);
    res.status(500).json({ message: 'Internal Server Error' });
  }
};
```

Frontend Implementation

The frontend handles the Google login popup and sends the ID token to the backend.

Google Login Using Firebase (React & Redux Toolkit)

Redux Thunk Action for Google Login

```
export const signInWithGoogle = createAsyncThunk('/google-login', async () => {
  try {
    // Open Google login popup
    const result = await signInWithPopup(auth, googleAuthProvider);

    // Get Firebase ID Token
    const idToken = await result.user.getIdToken();
    console.log(idToken);

    // Send ID Token to Backend API
    const res = await axios.post(
      `${import.meta.env.VITE_API_URL}/auth/google`,
      { idToken }
    );

    return res.data;
  } catch (err) {
    console.error('Google Sign-In Error:', err);
    throw err;
  }
});
```

Flow Diagram

[User Clicks "Sign in with Google"]

↓
[Google Authentication Popup Opens]
↓
[User Logs in with Google Account]
↓
[Firebase Issues ID Token]
↓
[Frontend Sends ID Token to Backend]
↓
[Backend Verifies ID Token with Firebase Admin SDK]
↓
[User Retrieved or Created in Database]
↓
[JWT Token Generated & Stored in Cookie]
↓
[User Successfully Authenticated]
