**Verify google signin token in the backend**

At the moment we only care about email from frontend to backend. Anyone can use postman and get access to our app using existing user's email. To prevent this we send token that we receive from google signin verify that token in the backend and if that returns true only then log the user in.

// React > SocialLogin.js

// only send tokenId

responseGoogle = response => {

// console.log('response', response);

const tokenId = response.tokenId;

const user = {

tokenId: tokenId

};

socialLogin(user).then(data => {

// console.log('signin data: ', data);

if (data.error) {

console.log('Error Login. Please try again..');

} else {

// console.log('signin success - setting jwt: ', data);

authenticate(data, () => {

console.log('social login response from api', data);

this.setState({ redirectToReferrer: true });

});

}

});

};

// now in backend verify that token

// if token is valid, you will get user info

// first add th env variable for client id

// which should be exactly same as it is in your react frontend's env > client id

// nodeapi

// .env

REACT\_APP\_GOOGLE\_CLIENT\_ID=xxxxxxxxxxxxxxx

// install npm package to verify google

npm i google-auth-library

// then your social-login controller method should be like so:

// require on top of controllers/auth

const { OAuth2Client } = require('google-auth-library');

// controllers/auth > socialLogin()

const client = new OAuth2Client(process.env.REACT\_APP\_GOOGLE\_CLIENT\_ID);

exports.googleLogin = async (req, res) => {

try {

const idToken = await req.body.tokenId;

const ticket = await client.verifyIdToken({ idToken, audience: process.env.REACT\_APP\_GOOGLE\_CLIENT\_ID });

// console.log('ticket', ticket);

const { email\_verified, email, name, picture, sub: googleid } = ticket.getPayload();

if (email\_verified) {

console.log(`email\_verified > ${email\_verified}`);

const newUser = { email, name, password: googleid };

// try signup by finding user with req.email

let user = User.findOne({ email }, (err, user) => {

if (err || !user) {

// create a new user and login

user = new User(newUser);

req.profile = user;

user.save();

// generate a token with user id and secret

const token = jwt.sign({ \_id: user.\_id, iss: process.env.APP\_NAME }, process.env.JWT\_SECRET);

res.cookie('t', token, { expire: new Date() + 9999 });

// return response with user and token to frontend client

const { \_id, name, email } = user;

return res.json({ token, user: { \_id, name, email } });

} else {

// update existing user with new social info and login

req.profile = user;

user = \_.extend(user, newUser);

user.updated = Date.now();

user.save();

// generate a token with user id and secret

const token = jwt.sign({ \_id: user.\_id, iss: process.env.APP\_NAME }, process.env.JWT\_SECRET);

res.cookie('t', token, { expire: new Date() + 9999 });

// return response with user and token to frontend client

const { \_id, name, email } = user;

return res.json({ token, user: { \_id, name, email } });

}

});

}

} catch (error) {

return res.json({

error: 'Request failed'

});

}

};