passport.serializeUser((user, done) => {

done(null, user.id); // Store user ID (or another unique identifier) in session

});

passport.deserializeUser(async (id, done) => {

try {

const user = await User.findById(id); // Retrieve user by ID from DB

done(null, user); // Attach user to `req.user`

} catch (err) {

done(err, null);

}

});

passport.deserializeUser(async (id, done) => {

console.log("Deserializing user ID:", id);

try {

const user = await User.findById(id);

if (!user) {

return done(new Error('User not found'));

}

done(null, user);

} catch (err) {

done(err, null);

}

});

Things I have done (suggested by ChatGPT):

Have this like this in index.js

app.use(cors({

origin: 'https://p2-web-services-w4.onrender.com', // Frontend URL

credentials: true, // Allow cookies and authorization headers

}));

Placed at top of index file as directed)

app.set('trust proxy', 1); // Trust the first proxy

I’ve verified this order:

app.use(session({ ... }));

app.use(passport.initialize());

app.use(passport.session());

Enforced HTTPS in production

if (process.env.NODE\_ENV === 'production') {

app.use((req, res, next) => {

if (req.headers['x-forwarded-proto'] !== 'https') {

return res.redirect(`https://${req.headers.host}${req.url}`);

}

next();

});

Does order matter here?

app.use(session({

  secret: 'victory-planner',

  resave: false,

  saveUninitialized: false,

  store: MongoStore.create({ mongoUrl: process.env.MONGODB\_URI }),

  cookie: {

    secure: process.env.NODE\_ENV === 'production', // Ensure cookies are only sent over HTTPS in production

    httpOnly: true,  // Prevents access to the cookie via JavaScript (XSS protection)

    sameSite: 'none',

    // maxAge: 24 \* 60 \* 60 \* 1000 // 1 day (adjust if needed)

  }

}));

Where do I put this in the index file?

app.use((req, res, next) => { console.log('Session:', req.session); next(); });

In what file would I put this?

fetch('https://your-backend.onrender.com', {

method: 'GET',

credentials: 'include', // Includes cookies

});

app.use((req, res, next) => {

console.log('Cookies:', req.cookies);

next();

});

app.use((req, res, next) => {

console.log('Headers:', req.headers);

next();

});

How would I:

Send requests with cookies and headers from Postman to isolate frontend issues.

**5. Authorization and Passport**

passport relies on sessions to maintain the user state. If the session isn’t working, req.isAuthenticated() will always return false.

Where do I put this:

app.use((req, res, next) => {

console.log('User:', req.user);

next();

});

the session cookie (connect.sid) is not being properly set during the login process, which is likely causing the authentication failure.

the session is being created successfully after the OAuth callback

the session cookie (connect.sid) is not being sent to the client. it's either not being set correctly or not being sent with the response after authentication.

the user information was not successfully retrieved and stored in the session.

the session is not being properly saved.

**5. Ensure Correct Domain for Cookies (I DON’T UNDERSTAND THIS)**

If your backend and frontend are on different domains or subdomains (e.g., frontend.example.com and api.example.com), you might need to specify the domain option in your cookie configuration so that the cookie is shared across subdomains.

For example:

js

Copy code

cookie: {

httpOnly: true,

secure: process.env.NODE\_ENV === 'production',

sameSite: 'none',

domain: '.example.com', // Ensure the cookie is valid across subdomains

}

**General Flow of Authentication in Your Setup:**

1. **User visits /auth/github route**: They are redirected to GitHub to authenticate.
2. **GitHub callback**: GitHub redirects the user back to your /auth/github/callback route.
3. **Passport Authentication**: passport.authenticate('github') handles the OAuth flow. If successful, Passport populates req.user and sets up the session.
4. **Session Cookie**: After authentication, req.session is populated with session data. The Set-Cookie header is sent to the client with the session ID.
5. **Logs**: You log req.user, req.session, and Set-Cookie to confirm everything is working.

From your earlier logs and the detailed req object you shared, the req object **does not yet have everything it needs**, as evidenced by these key observations:

1. **Missing Cookies:**
   * The connect.sid cookie is showing as undefined in your logs.
   * Cookies are critical for session-based authentication because they contain the session ID used to associate the user with their session in the server. Without the session cookie, the server cannot retrieve the user information from the session store.
2. **Missing Authorization Headers:**
   * In some of your logs, the authorization token (e.g., Bearer token) is also undefined, though this might not always be relevant if you're using sessions.
3. **Session Initialization:**
   * Your logs indicate that the session is being initialized (Session set after OAuth) and stored in the session store, but because the connect.sid cookie is not being sent back to the client or returned on subsequent requests, the session cannot be used to persist the authenticated state.

db.query(‘SELECT LAST\_INSERT\_ID() as user\_id’, function(error, results, fields) {

if (error) throw error;

console.log(results[0]);

req.login(results[0], function(err) {

res.redirect('/dashboard?registered=true')

});

});

const mongoose = require('mongoose'); // Assuming Mongoose is used

const User = require('../models/user'); // Adjust path to your User model

exports.loginUser\_id = async (req, res) => {

try {

// Find the most recently inserted user

const user\_id = await User.findOne().sort({ \_id: -1 }).lean();

if (!user\_id) {

return res.status(404).send({ message: 'No users found in the database.' });

}

console.log('Last inserted user:', user\_id);

// Perform login (req.login is usually part of Passport.js)

req.login(user\_id, (err) => {

if (err) {

console.error('Error during login:', err);

return res.status(500).send({ message: 'Error during login.' });

}

// Redirect upon successful login

res.redirect('/dashboard?registered=true');

});

} catch (error) {

console.error('Error retrieving the last inserted user:', error);

res.status(500).send({

message: 'Internal Server Error while retrieving the last inserted user.',

});

}

};

passport.serializeUser(function(user\_id, done) {

done(null, user\_id);

});

passport.deserializeUser(function(user\_id, done) {

done(err, user\_id);

});

**Option for bcrypt method use**

// Add `comparePassword` as a method userSchema.methods.comparePassword = async function (plainPassword) { return bcrypt.compare(plainPassword, this.password); };

Password.js LocalStrategy could use this code:

const isPasswordValid = await user.comparePassword(password);

Instead of:

const isPasswordValid = await comparePassword(password, user.password);

**Tweeks I could make:**

- Hbs.js/goBack: decide on code difference & effect – local verses production

- 401.hbs add flash.error like in layout/login.hbs

- Refine differences in Middleware/auth.js

Render Swagger doc: <https://event-planner-nkma.onrender.com/api-docs/>