Step 1:

Create a basic express app and listen on port 3000

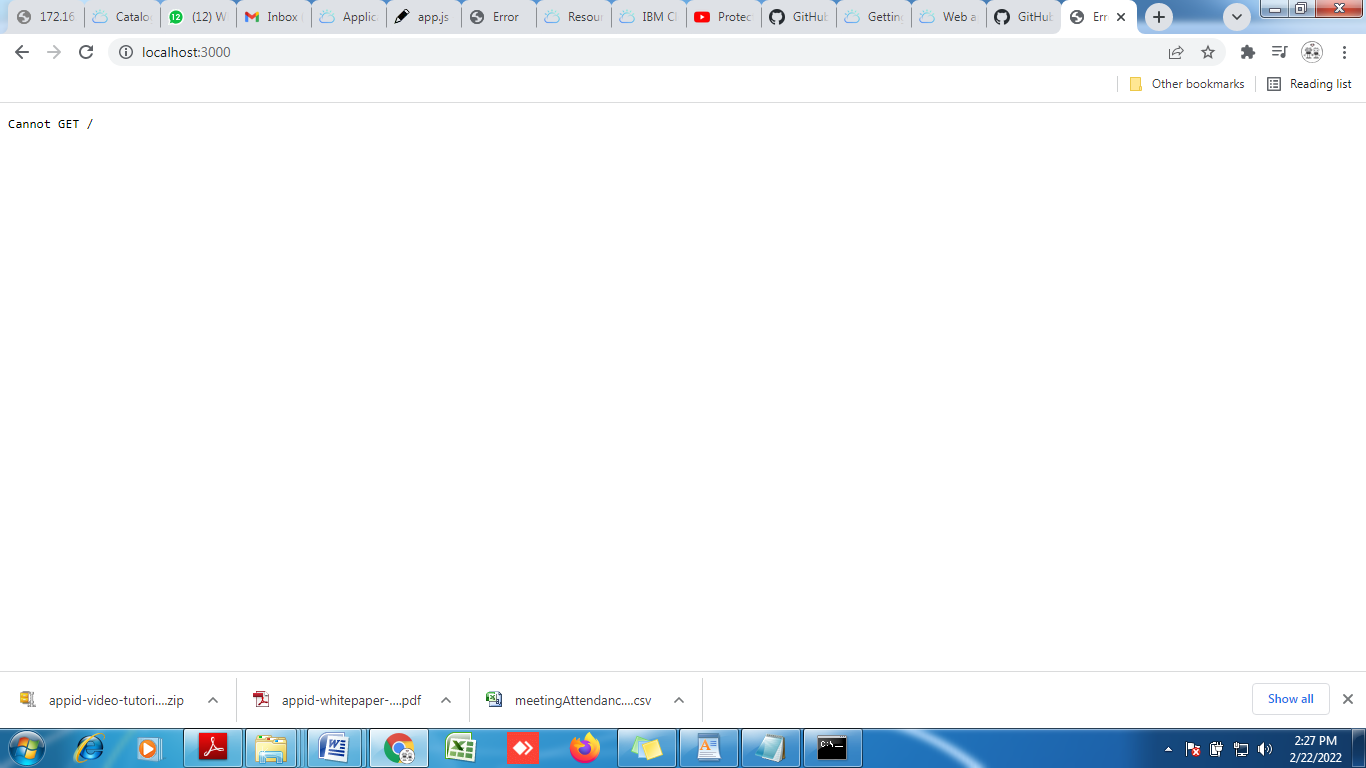
const express = require('express');

const app = express();

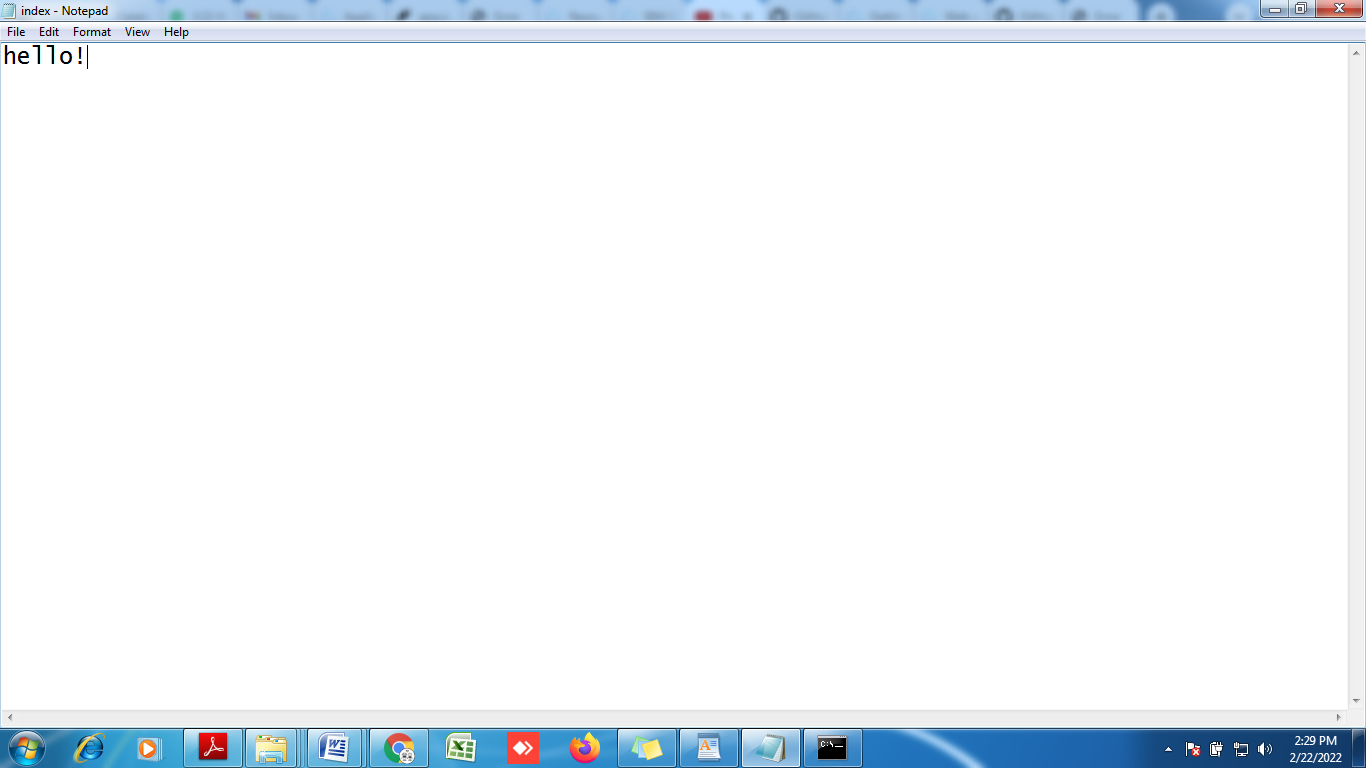
app.listen(3000, () => {

console.log('Listening on http://localhost:3000');

});



Create a folder with the name public and since it’s a web app so create a file index.html.



Now we update aap.js, to make it start serving the static sources.

const express = require('express');

const app = express();

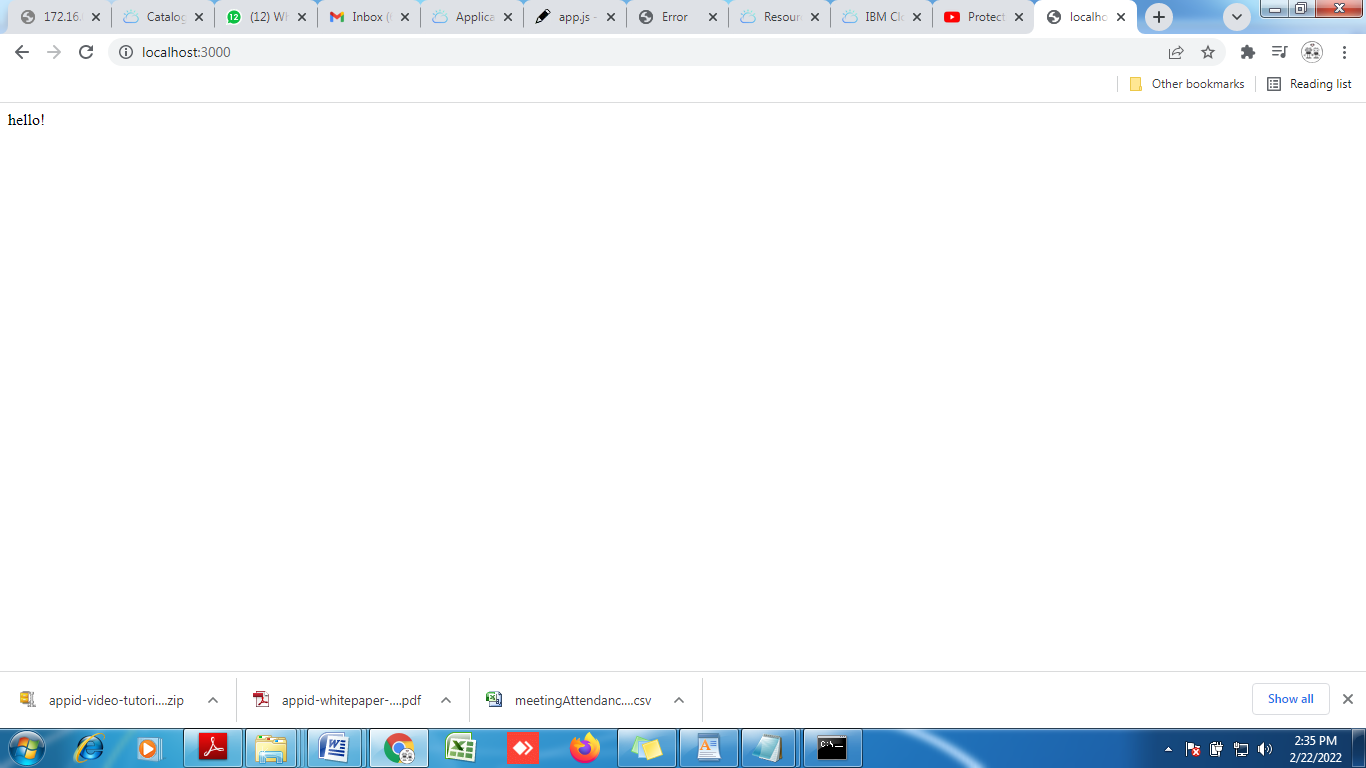
// Serve static resources

app.use(express.static('./public'));

app.listen(3000, () => {

console.log('Listening on http://localhost:3000');

});



Now make some changes in html file following syntax.

<!DOCTYPE html>

<html lang="en">

<head>

<title>Simple Node Web App</title>

</head>

<body>

<h1>Hello from the app!</h1>

</body>

</html>

After changes refresh the app.

At the moment this application is not protected. If we want to make sure that only the authenticated and authorized users are able to access this application. So. Lets start by adding the components of AppID.

Packages:

npm install express-session

npm install passport

npm install ibmcloud-appid

**ADD MODULES TO THE CODE**

const session = require('express-session');

const passport = require('passport');

const WebAppStrategy = require('ibmcloud-appid').WebAppStrategy;

**Now add some code that is required by the passport framework**

app.use(session({

secret: '123456',

resave: true,

saveUninitialized: true

}));

app.use(passport.initialize());

app.use(passport.session());

passport.serializeUser((user, cb) => cb(null, user));

passport.deserializeUser((user, cb) => cb(null, user));

**Next we need to tell passport to use the webappstarategy, that we just created And for this we need to configure the web app strategy , we need to provide the configuration that points towards AppID, and we need to have an APPID instance.**

* Go to IBM Cloud and create instance of APPID.
* Get the credentials by going to application section and create application.

**Add following code to app.js**

passport.use(new WebAppStrategy({

tenantId: "",

clientId: "",

secret: "",

oauthServerUrl: "",

redirectUri: "http://localhost:3000/appid/callback"

}));

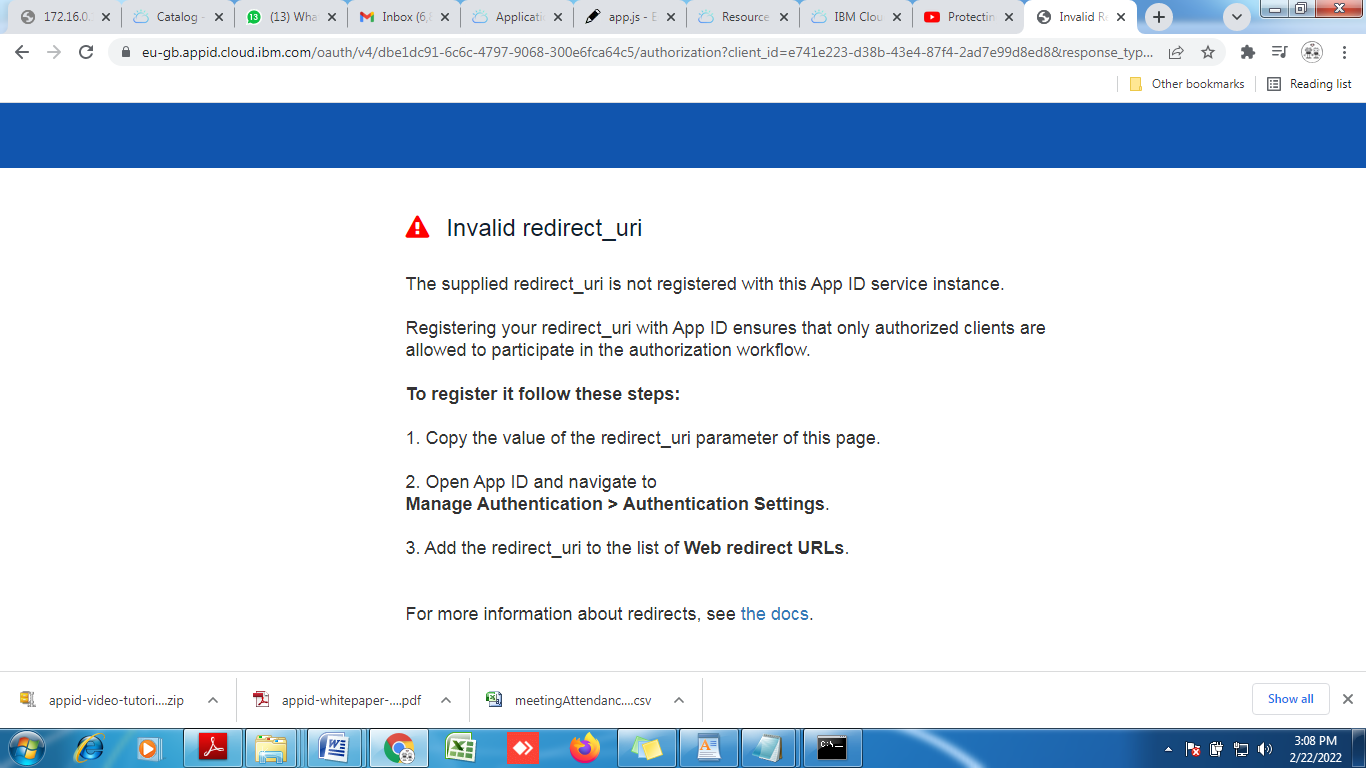
**After the successful authentication, app id is supposed to redirect you back to your application, for that purpose redirect uri is required.**

**Lets’s restart the application to check if everything is working fine.**

**Everything works fine. Now lets protect our application, by asking passport to use this web configuration for authentication.**

// Protect the whole app

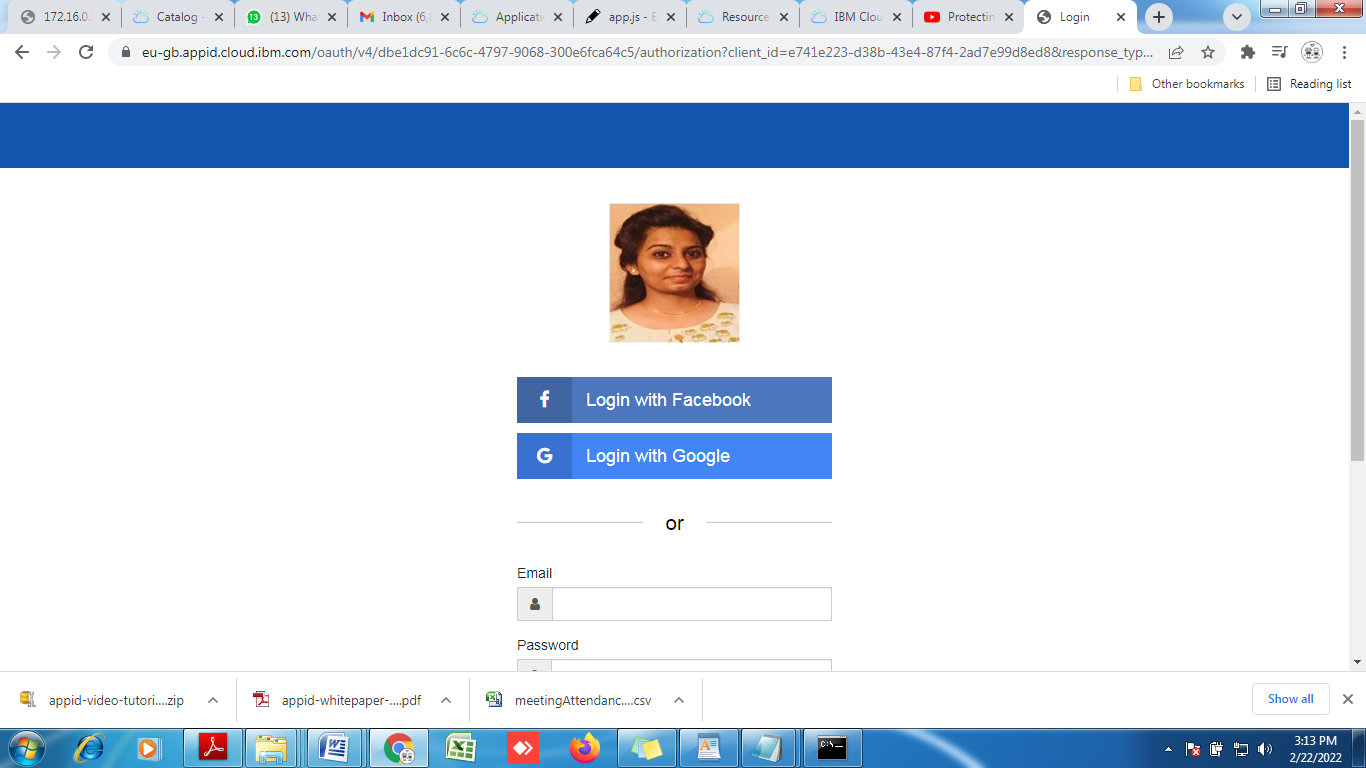
app.use(passport.authenticate(WebAppStrategy.STRATEGY\_NAME));



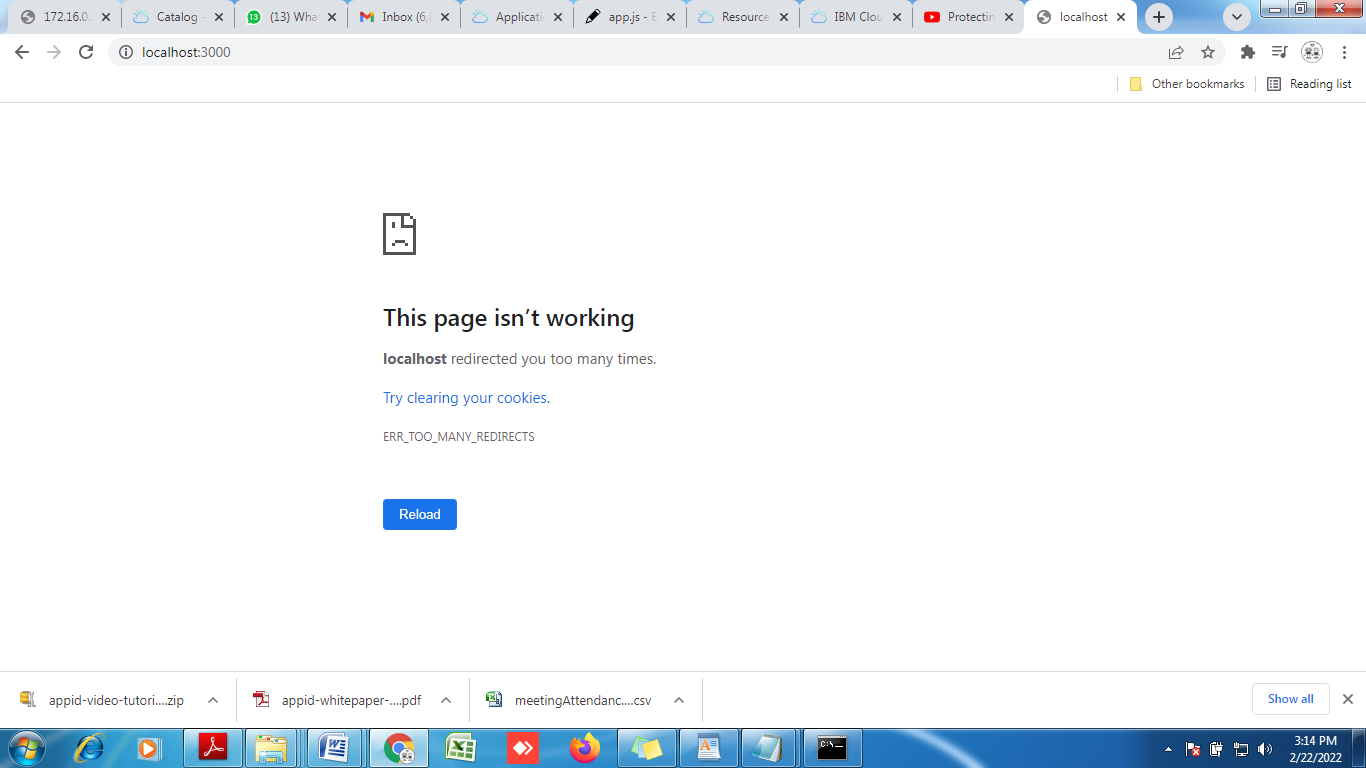
This is because before redirecting to the application we need to make sure that the uri is white-listed with AppID.

To fix this take the redirect URI and go to AppID and paste it in

>Manage Authentication> Authentication settings.



Now the issue is resolved., But there’s an another issue, when we try to login with google or facebook or whatso ever option . the output looks like this

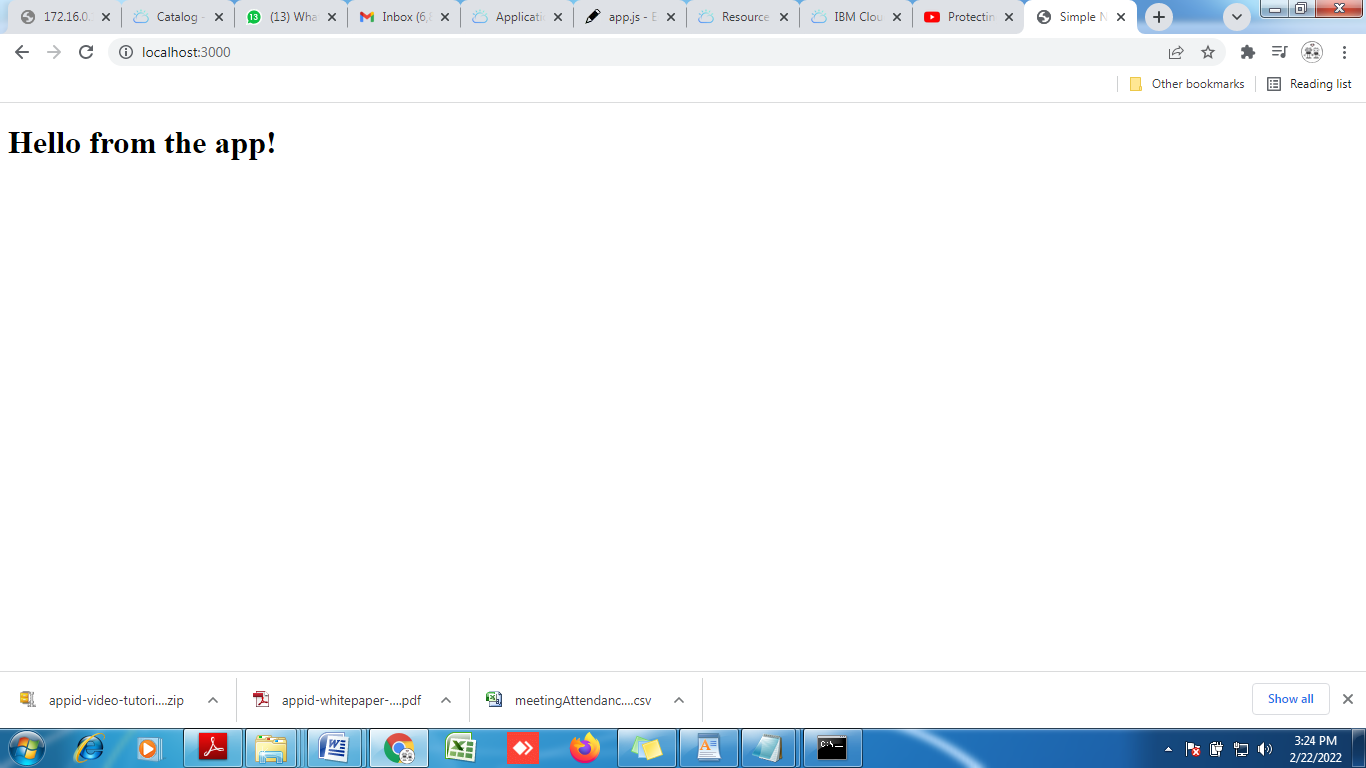


**This can be solved by handeling the appid callback before rest of the app is handeled.**

// Handle callback

app.get('/appid/callback', passport.authenticate(WebAppStrategy.STRATEGY\_NAME));

Restart the app



**Now we add the additional features. We add the logout part in to our application.To add that add the following code**

**app.get("/appid/logout", function(req, res, next) {**

**req.logout(function(err) {**

**if (err) { return next(err); }**

**res.redirect("/");**

**});**

**});**

**Also it would be great if we add some logout button. For that go back to index.html and add button part.**

<!DOCTYPE html>

<html lang="en">

<head>

<title>Simple Node Web App</title>

</head>

<body>

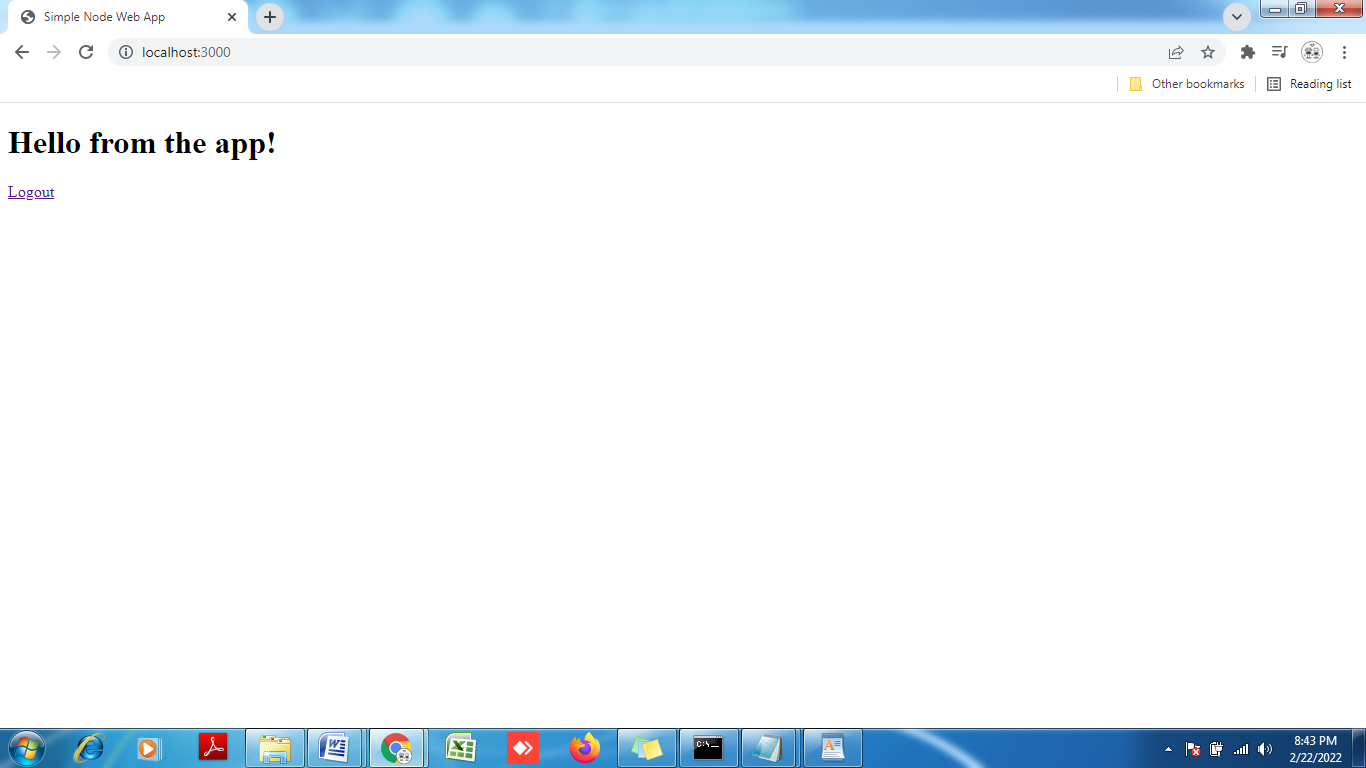
<h1>Hello from the app!</h1>

<a href="/appid/logout" id="logout">Logout</a>

</body>

</html>

**Output:**

****

**Taking one step further, what if I do not want to protect the whole application?**

**What if I want to protect only a particular section of my application.**

**So , to do that remove or comment this part that is protecting the full application.**

**Also we will add one more endpoint that will be for login.**

app.get('/appid/login', passport.authenticate(WebAppStrategy.STRATEGY\_NAME, {

successRedirect: '/',

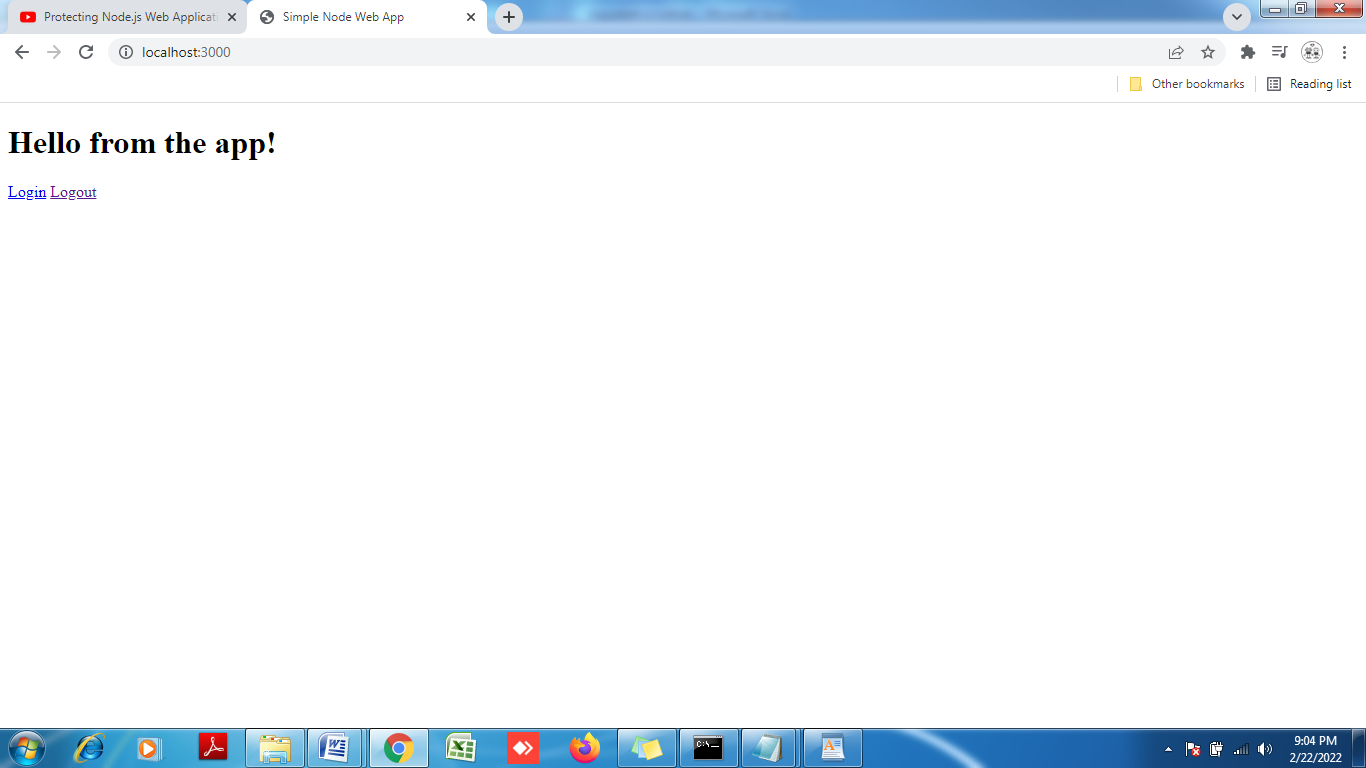
forceLogin: true

}));

**Also similar to logout, we will have login button as well. Add it in html file.**

**<a href="/appid/login" id="login">Login</a> /// add this in body**

**Now restart the application.**

****

**Question here is that, we are authenticated but how do we get to know that we are authenticated. There is nothing on the screen that shows that we are authenticated.**

**Can application present any kind of information about the user?**

**And the answer is obviously yes. To get the information about user we need to add some code in html and to retrieve information about user from server and bring it on browser we add some code in app.js.**

**Add the following line in html file**

<h3 id="user"></h3> // below logout.

**Add the following code in app.js. we create an API end point for the user information retrieval.**

**// The /api/user API used to retrieve name of a currently logged in user**

app.get('/api/user', (req, res) => {

// console.log(req.session[WebAppStrategy.AUTH\_CONTEXT]);

res.json({

user: {

name: req.user.name

}

});

});

**Also we will add some script in html to handle the user information on successful authentication.**

<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js"></script>

<script>

$().ready(() => {

$.get("/api/user")

.then((resp) => {

$("#user").text("Hello " + resp.user.name);

$("#logout").show();

$("#login").hide();

});

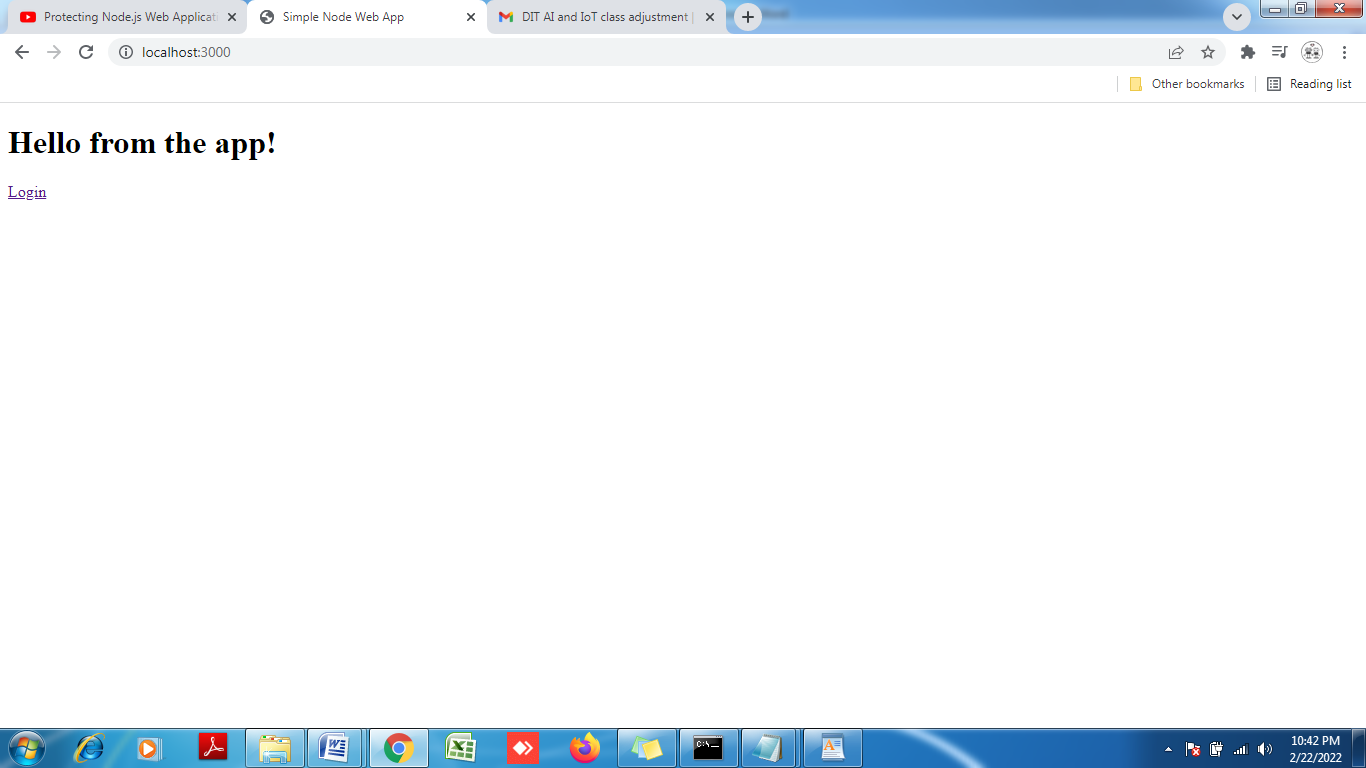
});

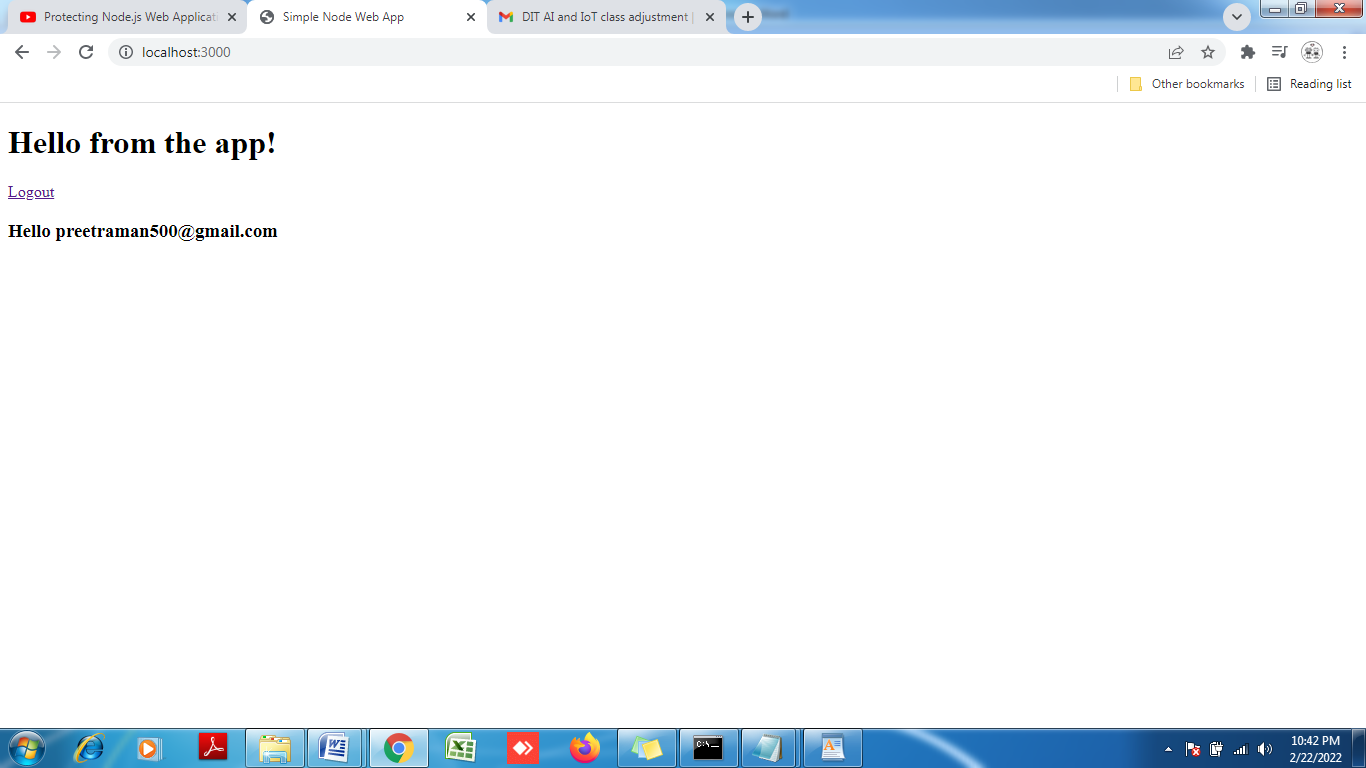
</script>

**Also we hide the logout button by default.**

<a href="/appid/logout" id="logout" style="display: none;">Logout</a>

**After this lets test our application what happens. Lets refresh our application.**

****

****

**Additional thing we can do is only when the authenticated user name is available, only then collect the information , else do not call the user response part.**

**Add the following code:**

**app.use('/api', (req, res, next) => {**

**if (req.user){**

**next();**

**} else {**

**res.status(401).send("Unauthorized");**

**}**

**});**

**// The /api/user API used to retrieve name of a currently logged in user**

**app.get('/api/user', (req, res) => {**

**// console.log(req.session[WebAppStrategy.AUTH\_CONTEXT]);**

**res.json({**

**user: {**

**name: req.user.name**

**}**

**});**

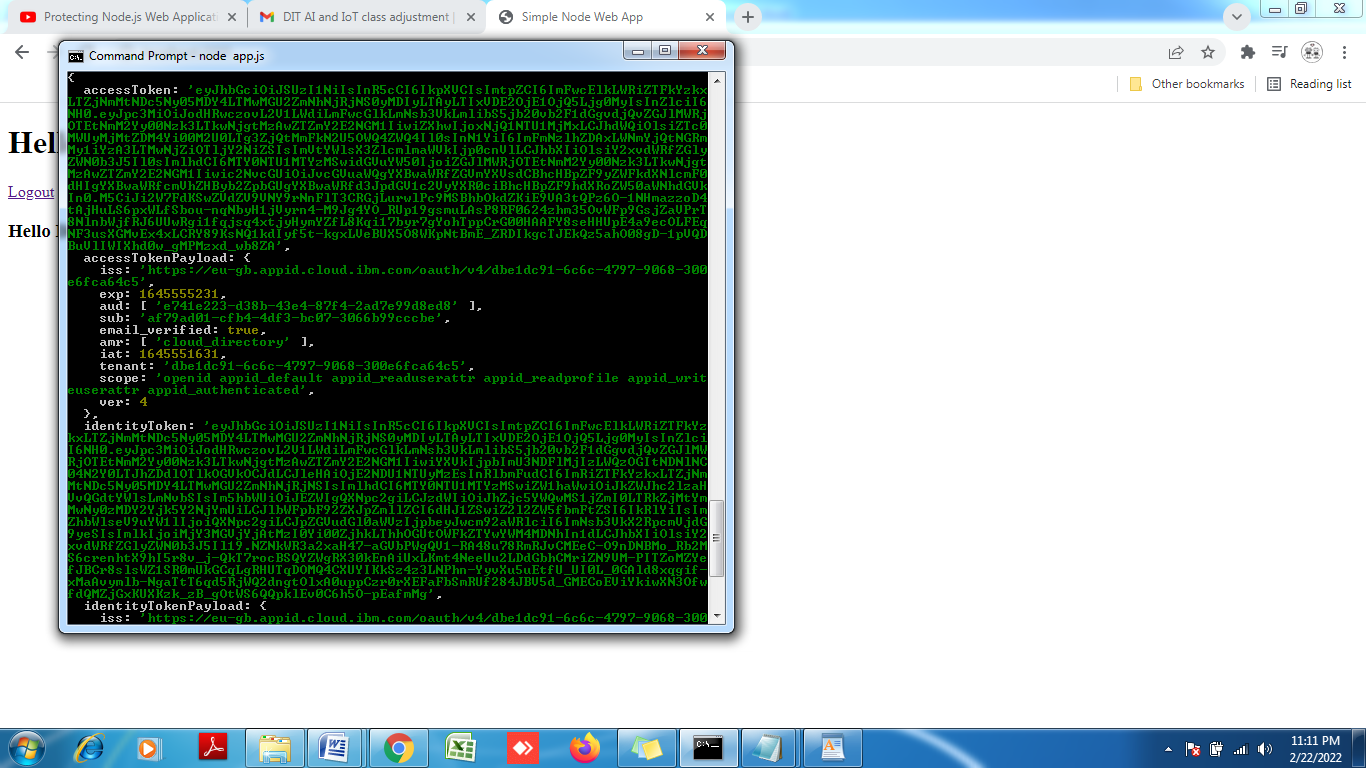
**});**

**Now the part we are left with is how to work with tokens.**

**AppID not only provides information about user name, but it also gives information about user access level, so we add the following line to the code**

console.log(req.session[WebAppStrategy.AUTH\_CONTEXT]);

this line prints the current sessions AUTHENTICATION CONTEXT.

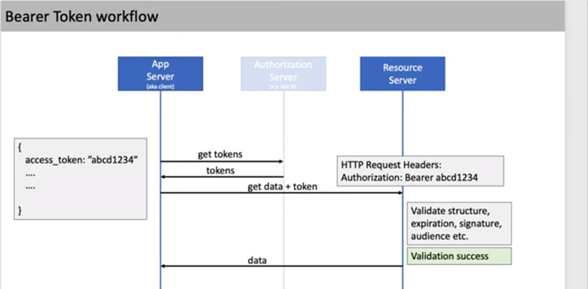


Jwt.io /// to decode json web token.

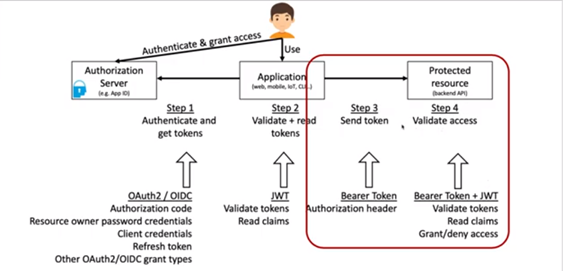
Try same thing on cloud.

Protecting Node js Backend App:

Here we are going to see how actually the bearer token workflow.



Here e see step 3 and step 4



First create a simple node js app

As previous, first create a basic express app running on port 3000

const express = require('express');

const app = express();

// The /api/data API used to retrieve

app.get('/api/data', (req, res) => {

res.json({

data: 12345

});

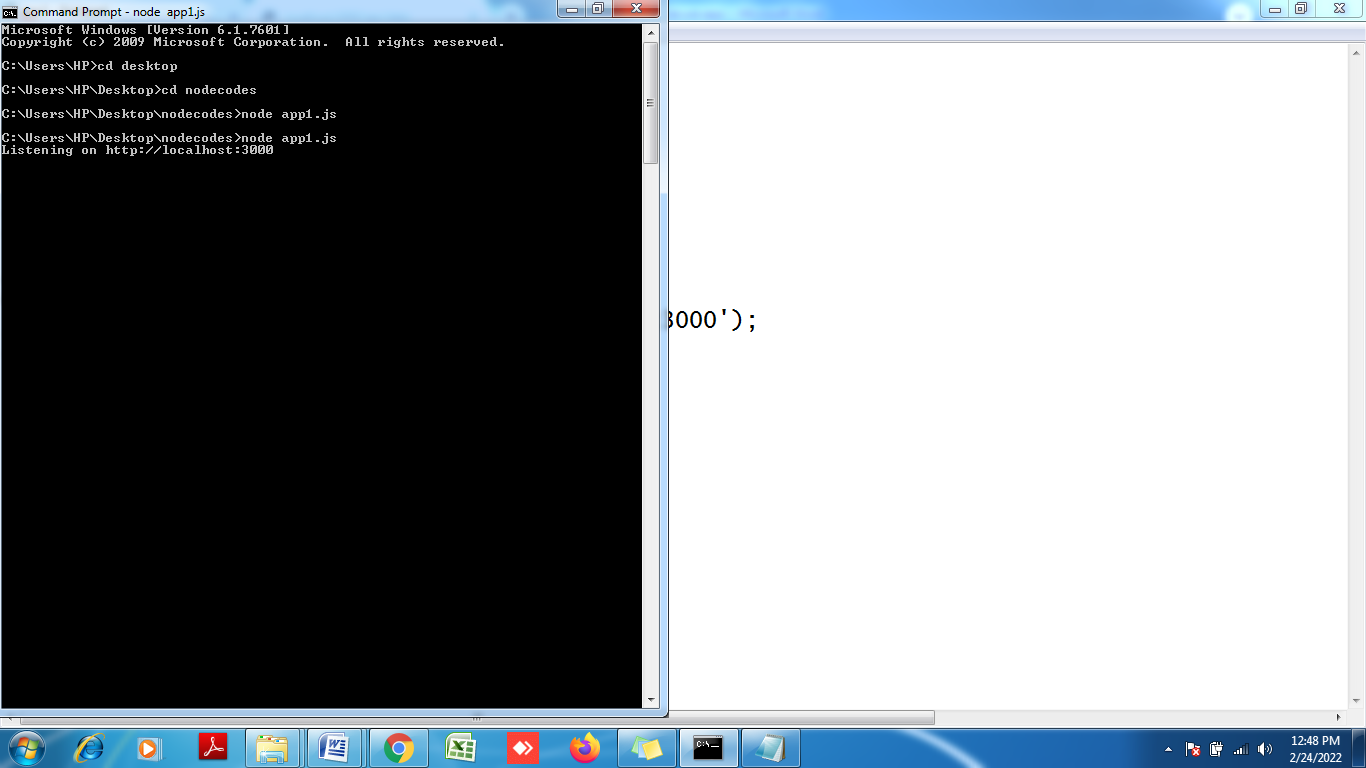
});

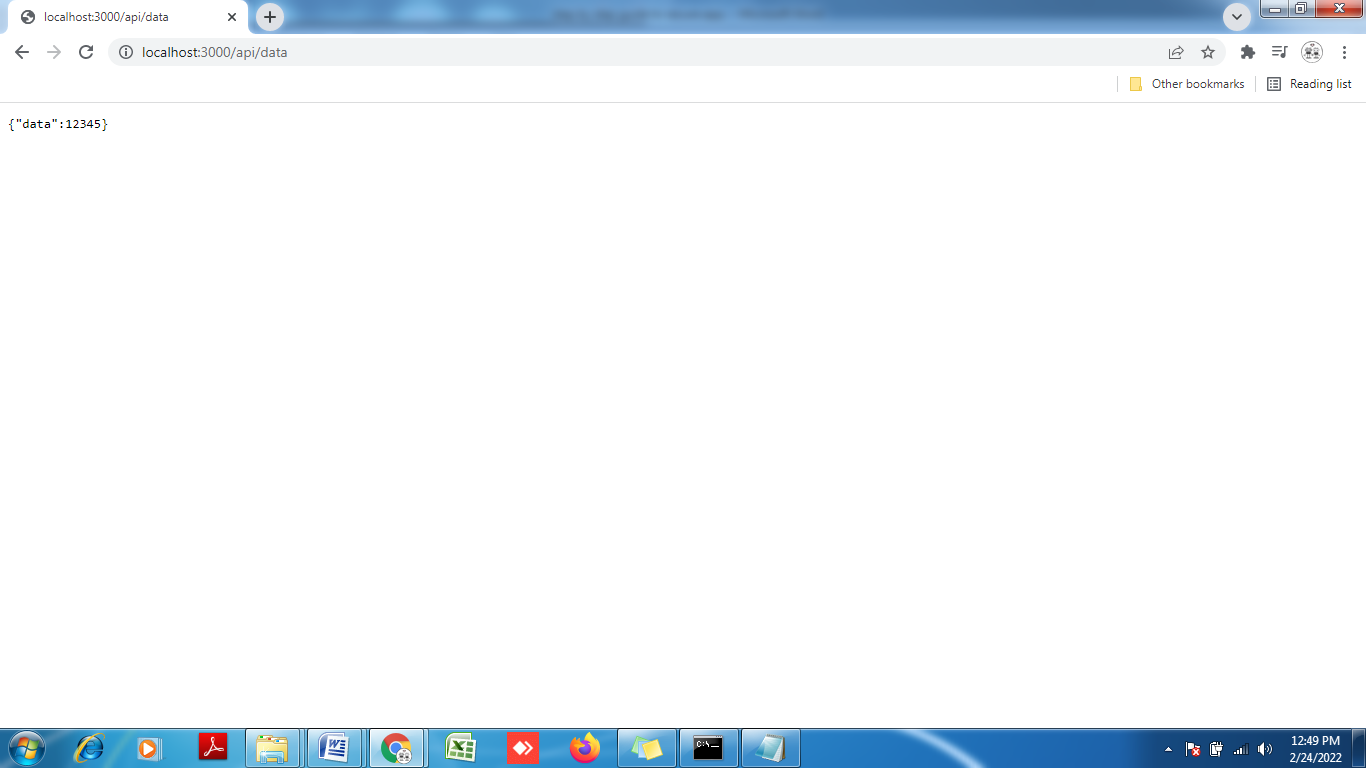
app.listen(3000, () => {

console.log('Listening on http://localhost:3000');

});

Run the application.





Nov adding code to protect the Backend API data.

const express = require('express');

const passport = require('passport');

const APIStrategy = require('ibmcloud-appid').APIStrategy;

const app = express();

app.use(passport.initialize());

passport.use(new APIStrategy({

oauthServerUrl: "https://eu-gb.appid.cloud.ibm.com/oauth/v4/dbe1dc91-6c6c-4797-9068-300e6fca64c5",

}));

// Protect the whole app

app.use(passport.authenticate(APIStrategy.STRATEGY\_NAME, {

session: false

}));

// The /api/data API used to retrieve protected data

app.get('/api/data', (req, res) => {

res.json({

data: 12345

});

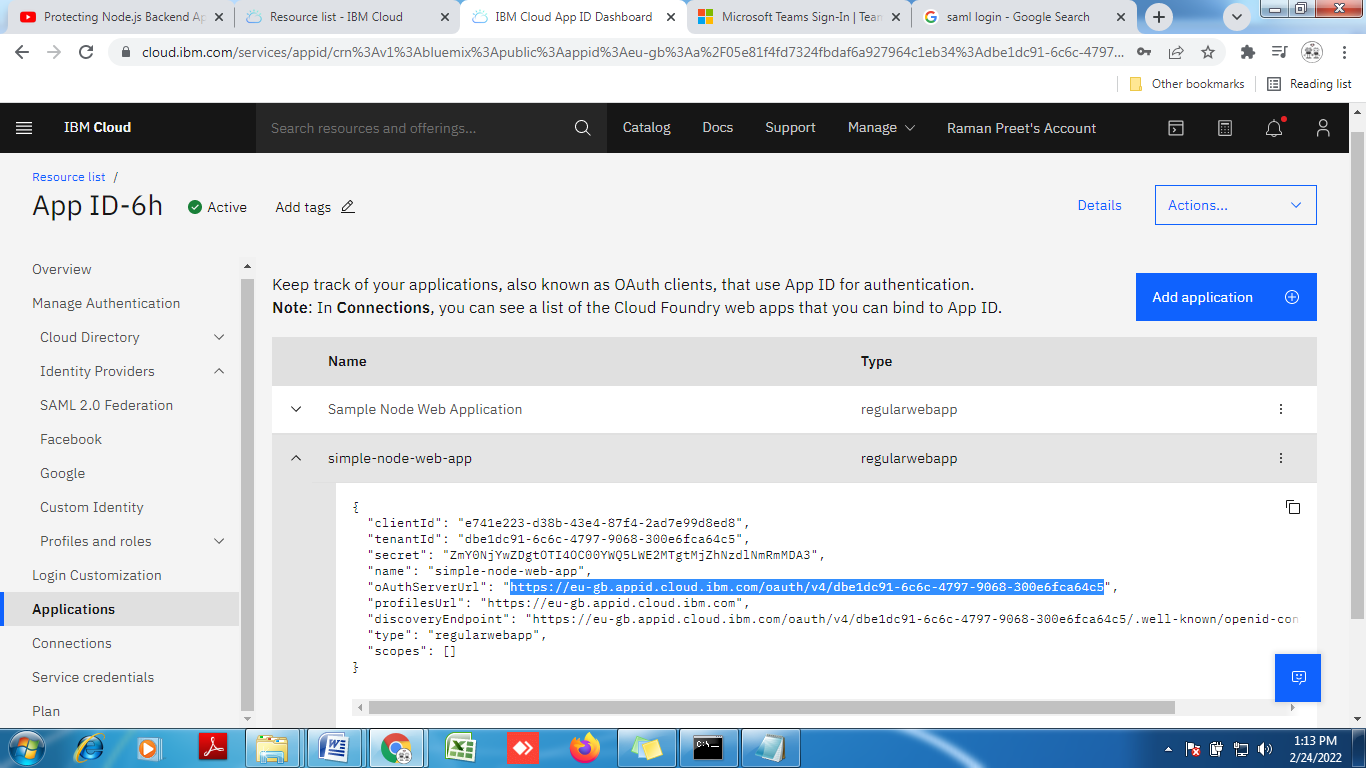
});

app.listen(3000, () => {

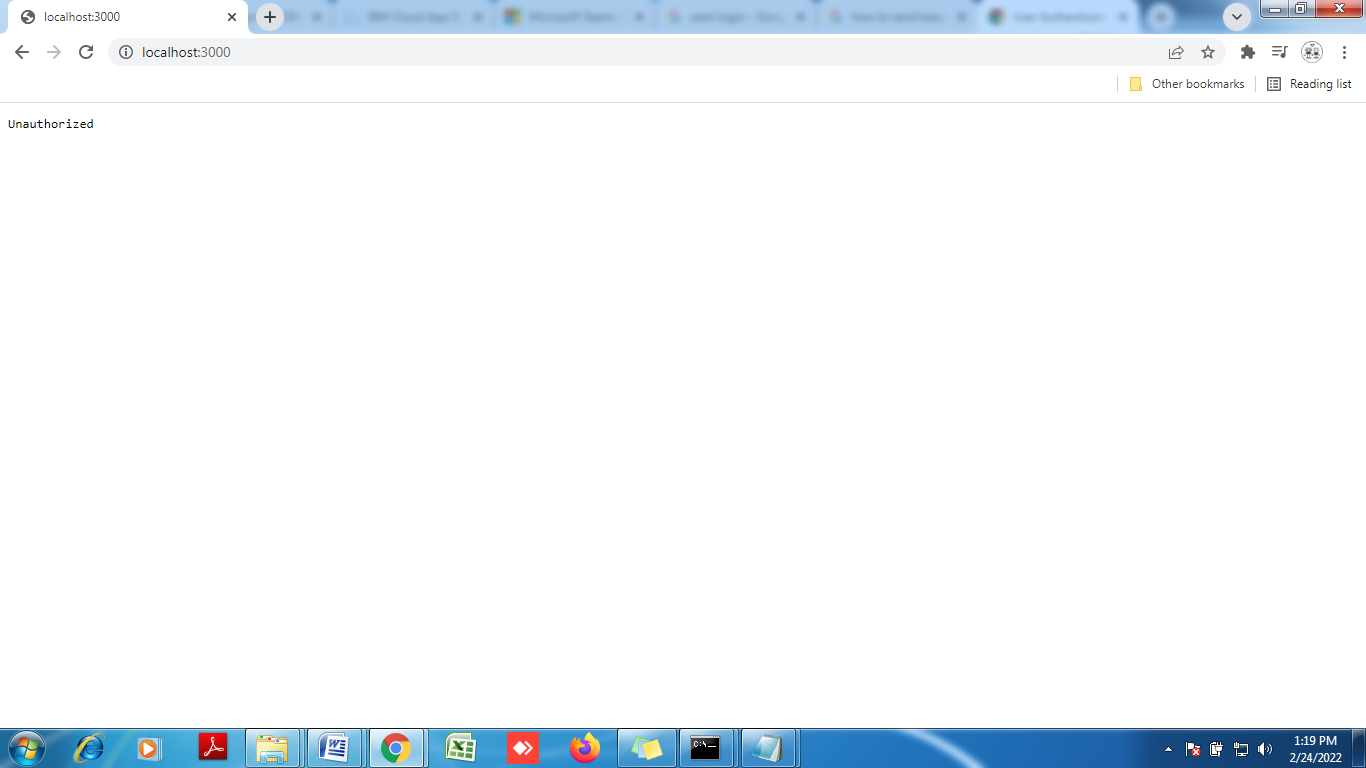
console.log('Listening on http://localhost:3000');

});

Get the oauthserverUrl from AppID



OUTPUT:



We need to provide the access token.

Attach the bearer token.