

Name - Yash Lakhtariya  
Enrollment number - 21162101012  
Branch - CBA      Batch - 71  
CS Practical 12

**Aim :** You are tasked with setting up a secure authentication system for an enterprise application hosted on IBM Cloud. The application should enforce MFA for added security and integrate with a custom identity provider to accommodate users from a legacy system. Also work on the process of regenerating access tokens using refresh tokens.

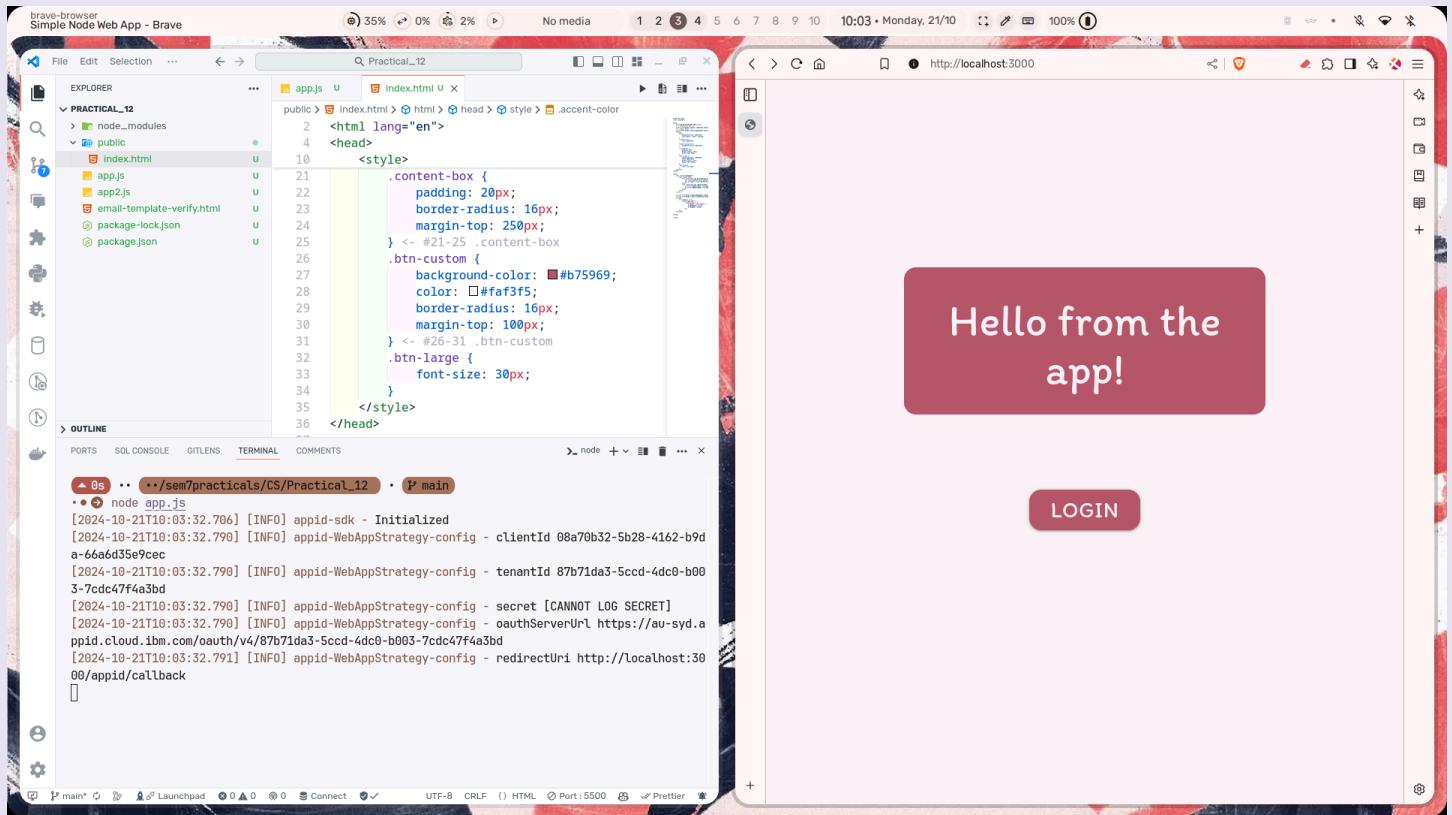
### Steps and Screenshots:

#### 1. Enable MFA from Cloud directory settings of App ID instance

The screenshot shows the IBM Cloud App ID Dashboard in a browser window. The URL is https://cloud.ibm.com/appid?panelId=cloud\_directory\_multifactor&tenantId=cm%3Av1%3Abluemix%3Apublic%3Appid%3Aau-syd%3Aa%2F9553f7184dd92a056f240cf78ef6%3A87b71da3-5cc4-4dc0-b003-7cd47f4a3bd%3A%3... . The dashboard has a sidebar on the left with options like Workspaces, Home, ML, CD, CCE, CS, IoT, Start Page, and Identity & Access Management. The main area shows a resource list for 'yashlani-appid' which is active. Under the 'Multifactor authentication (MFA)' section, the 'Settings' tab is selected. A red box highlights the 'Enable multifactor authentication (MFA)' toggle switch, which is set to 'Enabled'. Below it, under 'Authentication method', there is a radio button for 'Email' which is selected, and another for 'Text messages (SMS)'. The status bar at the bottom shows the date as Thursday, 24/10 and the time as 11:39 AM.

Name - Yash Lakhtariya  
Enrollment number - 21162101012  
Branch - CBA      Batch - 71  
CS Practical 12

## 2. Check if MFA works



Name - Yash Lakhtariya  
Enrollment number - 21162101012  
Branch - CBA      Batch - 71  
CS Practical 12

The screenshot shows a Brave browser window with two tabs. The left tab is titled 'Practical\_12' and displays the code for a Node.js application. The right tab is titled 'Multi-Factor Authentication - Brave' and shows a step-by-step guide for setting up MFA. It includes instructions about preventing account access if the password is compromised, a message from IBM AppID, and a 'Continue' button.

```
public > index.html > html > head > style > accent-color
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4    <title>Simple Node Web App</title>
5    <!-- Materialize CSS -->
6    <link href="https://cdnjs.cloudflare.com/ajax/libs/materialize/1.0.0/css/materialize.min.css" type="text/css" media="screen,projection" rel="stylesheet"/>
7    <!-- Google Fonts -->
8    <link href="https://fonts.googleapis.com/css?family=Itim&display=block" type="text/css" media="screen,projection" rel="stylesheet"/>
9    <style>
10   body {
11     background-color: #faf3f5;
12     font-family: 'Itim', cursive;
13   }
14   .accent-color {
15     color: #faf3f5;
16   }
17   .accent-background {
18     background-color: #b75969;
19   }

```

```
node app.js
[2024-10-21T09:57:54.498] [INFO] appid-sdk - Initialized
[2024-10-21T09:57:54.620] [INFO] appid-WebAppStrategy-config - clientId 08a70b32-5b28-4162-b9d
a-66a6d35e9cec
[2024-10-21T09:57:54.620] [INFO] appid-WebAppStrategy-config - tenantId 87b71da3-5ccd-4dc0-b00
3-7cdc47f4a3bd
[2024-10-21T09:57:54.620] [INFO] appid-WebAppStrategy-config - secret [CANNOT LOG SECRET]
[2024-10-21T09:57:54.620] [INFO] appid-WebAppStrategy-config - oauthServerUrl https://au-syd.a
ppid.cloud.ibm.com/oauth/v4/87b71da3-5ccd-4dc0-b003-7cdc47f4a3bd
[2024-10-21T09:57:54.621] [INFO] appid-WebAppStrategy-config - redirectUri http://localhost:30
00/appid/callback
```

The screenshot shows a Brave browser window with two tabs. The left tab is titled 'Practical\_12' and displays the code for a simple Node.js application. The right tab is titled 'Simple Node Web App - Brave' and shows a red box containing the text 'Hello from the app!' and 'Hello Yash Lakhtariya'. A 'LOGOUT' button is also visible.

```
identityTokenPayload: {
  iss: 'https://au-syd.appid.cloud.ibm.com/oauth/v4/87b71da3-5ccd-4dc0-b003-7cdc47f4a3bd',
  aud: [ '08a70b32-5b28-4162-b9d-a-66a6d35e9cec' ],
  exp: 172948927,
  tenant: '87b71da3-5ccd-4dc0-b003-7cdc47f4a3bd',
  iat: 1729485327,
  email: 'yash.lakhtariya21@gnu.ac.in',
  name: 'Yash Lakhtariya',
  sub: '710c6908-0e4a-4e27-9a99-7c1318f542d5',
  email_verified: true,
  given_name: 'Yash',
  family_name: 'Lakhtariya',
  identities: [ [Object] ],
  amr: [ 'cloud_directory', 'mfa_email' ],
  ver: 4
}
```

Name - Yash Lakhtariya  
Enrollment number - 21162101012  
Branch - CBA      Batch - 71  
CS Practical 12

### 3. Generate ssl keys to use with custom identity provider

The screenshot shows the IBM Cloud App ID Dashboard within a Vivaldi browser window. The main title is 'Manage identity providers'. On the left sidebar, under 'Manage Authentication', there are several options: Cloud Directory, Identity Providers (selected), Profiles and roles, Login Customization, Applications, Service credentials, and Plan. The 'Identity providers' tab is active. It lists several providers: Cloud Directory (Enabled), SAML 2.0 Federation (Disabled), Facebook (Enabled), Google (Enabled), Custom Identity (Enabled, highlighted with a red box), and Anonymous (Enabled). A success message at the top right says 'Changes saved successfully'. The URL in the address bar is https://cloud.ibm.com/appid?panel=IDP\_Configuration&tenantId=cm%3Av1%3Abluemix%3Apublic%3Appid%3Aau-syd%3Aa%2F955f5f7184ddb922a056f240c778ef6%3A87b71da3-5ccd-4dc0-b003-7cd47f4a3bd%3A%3A#/IDP\_Co...

Name - Yash Lakhtariya  
Enrollment number - 21162101012  
Branch - CBA      Batch - 71  
CS Practical 12

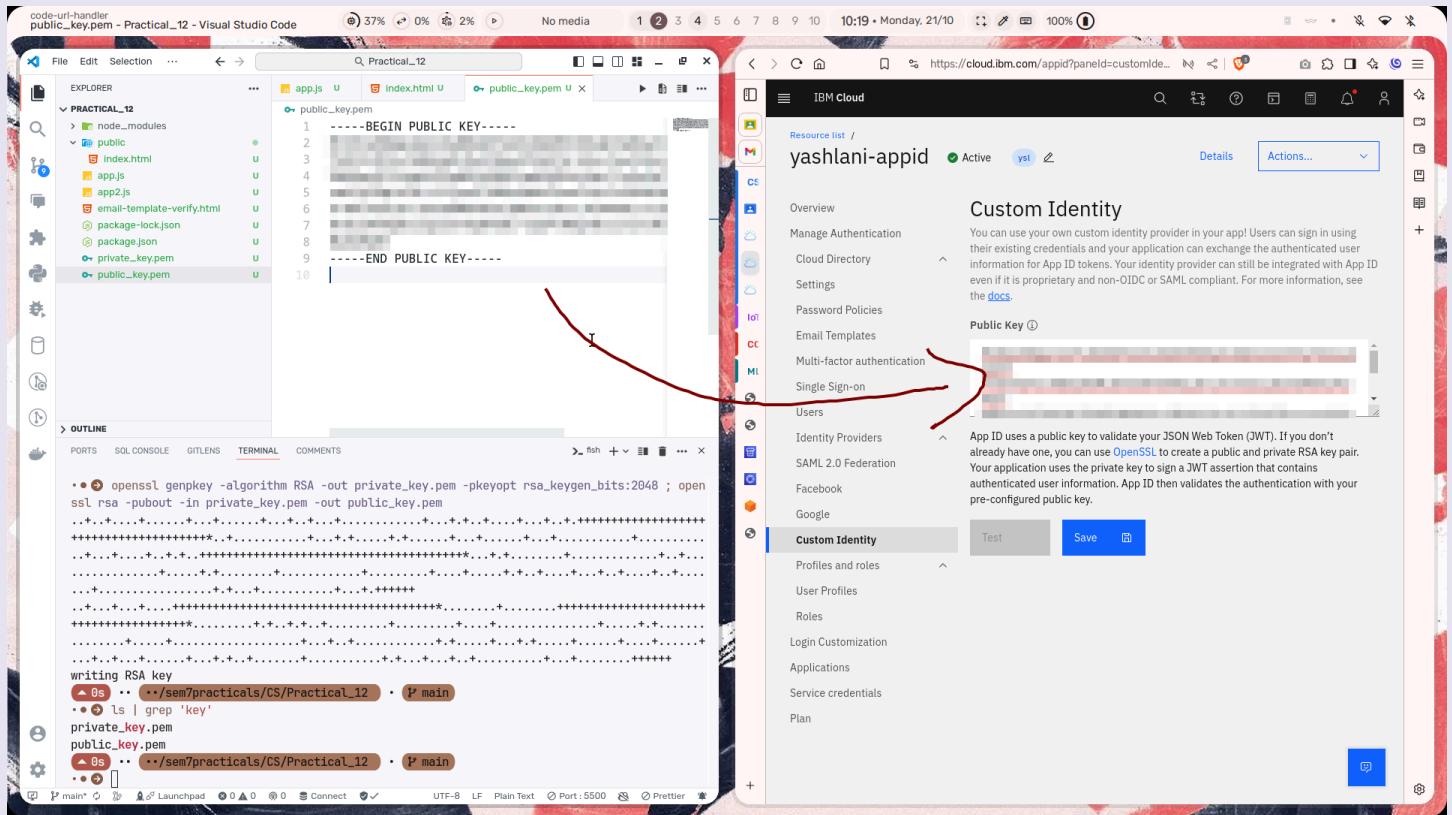
The screenshot shows the Visual Studio Code interface with the following details:

- EXPLORER:** Shows the project structure under "PRACTICAL\_12". The "public" folder contains "index.html", "app.js", "app2.js", "email-template-verify.html", "package-lock.json", "package.json", "private\_key.pem", and "public\_key.pem".
- CODE EDITOR:** The "index.html" file is open. The code includes HTML, CSS, and JavaScript. Key parts of the code are:

```
<html lang="en">
  <body>
    <div class="container">
      <div class="row">
        <div class="col s12 m8 offset-m2 l6 offset-l3 accent-background content-box center-align">
          <h1 class="accent-color">Hello from the app!</h1>
          <h3 id="user" class="accent-color"></h3>
        </div>
        <div class="col s12 center-align">
          <a href="/appid/Login" id="login" class="btn btn-custom btn-large">Login</a>
          <a href="/appid/logout" id="logout" class="btn btn-custom btn-large" style="display: none;">Logout</a>
        </div>
      </div>
    </div>
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js"></script>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/materialize/1.0.0/js/materialize.min.js"></script>
    <script>
      $(document).ready(() => {
```
- TERMINAL:** Shows command-line history related to RSA key generation and file listing.
- STATUS BAR:** Shows file statistics (Ln 48, Col 19), workspace settings (Spaces: 4, UTF-8, CRLF), port information (Port: 5500), and the Prettier extension icon.

Name - Yash Lakhtariya  
Enrollment number - 21162101012  
Branch - CBA      Batch - 71  
CS Practical 12

#### 4. Paste in App ID custom identity provider, the public key generated



Name - Yash Lakhtariya  
Enrollment number - 21162101012  
Branch - CBA      Batch - 71  
CS Practical 12

Brave-browser  
IBM Cloud App ID Dashboard - Brave

Resource list / yashlani-appid Active ysl

IBM Cloud Catalog Manage 2716063 - IBM India Pvt Ltd, C/o ... Details Actions...

Custom Identity

You can use your own custom identity provider in your app! Users can sign in using their existing credentials and your application can exchange the authenticated user information for App ID tokens. Your identity provider can still be integrated with App ID even if it is proprietary and non-OIDC or SAML compliant. For more information, see the [docs](#).

Public Key

-----END PUBLIC KEY-----

App ID uses a public key to validate your JSON Web Token (JWT). If you don't already have one, you can use [OpenSSL](#) to create a public and private RSA key pair. Your application uses the private key to sign a JWT assertion that contains authenticated user information. App ID then validates the authentication with your pre-configured public key.

Test Save

Success! Changes saved successfully

Overview Manage Authentication Cloud Directory Settings Password Policies Email Templates Multi-factor authentication Single Sign-on Users Identity Providers SAML 2.0 Federation Facebook Google Custom Identity Profiles and roles User Profiles Roles Login Customization Applications Service credentials Plan

Name - Yash Lakhtariya  
Enrollment number - 21162101012  
Branch - CBA      Batch - 71  
CS Practical 12

### 5. Use nodejs app to generate JWT and test it on IBM Cloud

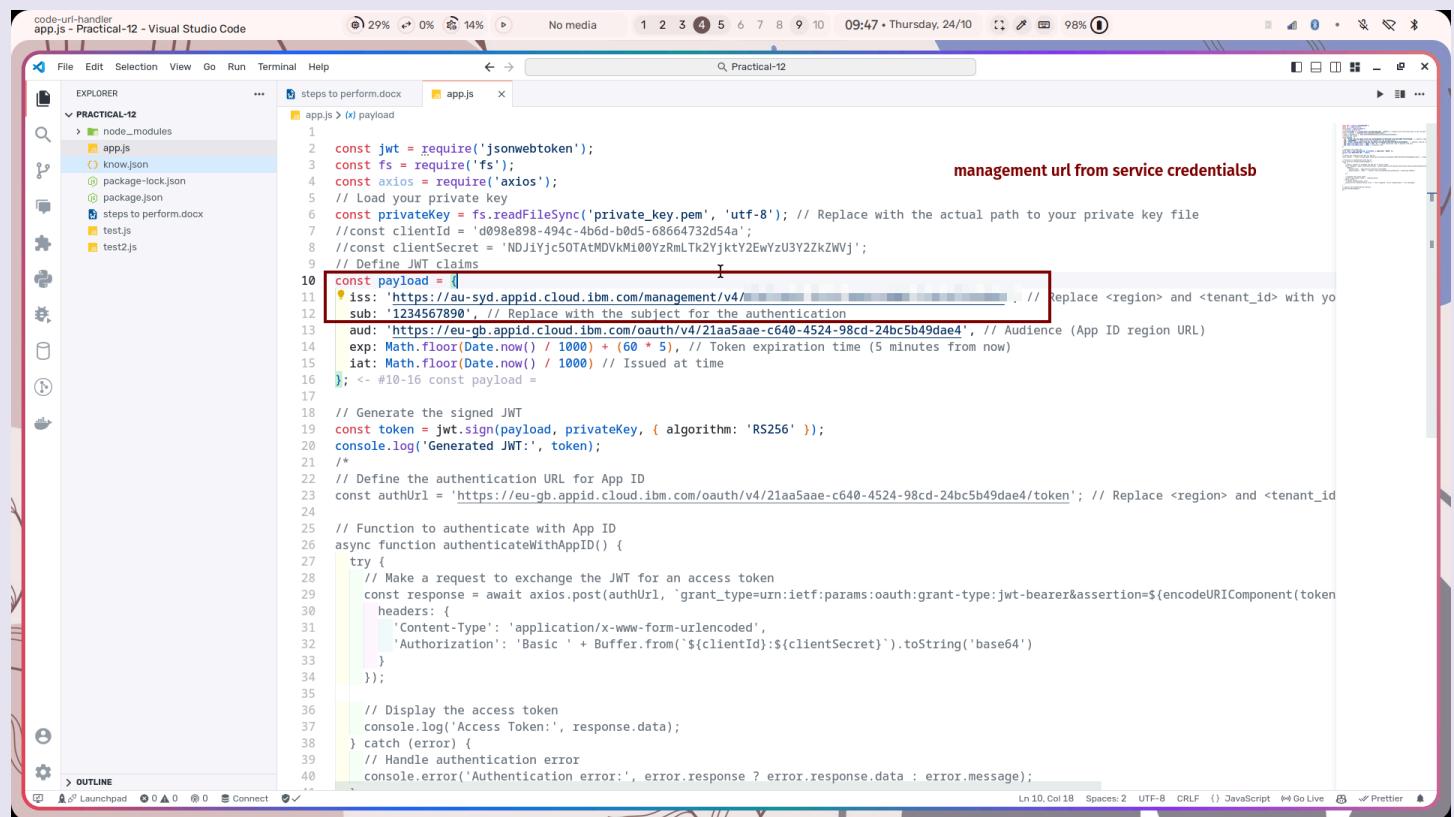
Code :

```
const jwt = require('jsonwebtoken');
const fs = require('fs');
// const axios = require('axios');
// Load your private key
const privateKey = fs.readFileSync('private_key.pem', 'utf-8');
// Replace with the actual path to your private key file
//const clientId = 'd098e898-494c-4b6d-b0d5-68664732d54a';
//const clientSecret =
'NDJiYjc50TAtMDVkMi00YzRmLTk2YjktY2EwYzU3Y2ZkZWVj';
// Define JWT claims
const payload = {
  iss:
'https://au-syd.appid.cloud.ibm.com/management/v4/xxxxxxxxx', // Replace <region> and <tenant_id> with your App ID region and tenant ID
  sub: 'Yashlani access token maate', // Replace with the subject for the authentication
  aud: 'https://au-syd.appid.cloud.ibm.com/oauth/v4/xxxxxxxxx',
// Audience (App ID region URL)
  exp: Math.floor(Date.now() / 1000) + (60 * 2), // Token expiration time (20 minutes from now)
  iat: Math.floor(Date.now() / 1000) // Issued at time
```

Name - Yash Lakhtariya  
Enrollment number - 21162101012  
Branch - CBA      Batch - 71  
CS Practical 12

};

```
// Generate the signed JWT
const token = jwt.sign(payload, privateKey, { algorithm: 'RS256' });
console.log('Generated JWT:', token);
```



The screenshot shows the Visual Studio Code interface with the file 'app.js' open. The code is as follows:

```
code-url-handler
app.js - Practical-12 - Visual Studio Code
File Edit Selection View Go Run Terminal Help
No media 1 2 3 4 5 6 7 8 9 10 09:47 • Thursday, 24/10 98% ⓘ
Practical-12
EXPLORER
PRACTICAL-12
node_modules
app.js
know.json
package-lock.json
package.json
steps to perform.docx
test.js
test2.js
app.js > [x] payload
1 const jwt = require('jsonwebtoken');
2 const fs = require('fs');
3 const axios = require('axios');
4 // Load your private key
5 const privateKey = fs.readFileSync('private_key.pem', 'utf-8'); // Replace with the actual path to your private key file
6 //const clientId = 'd098e898-494c-4b6d-b0d5-68664732d54a';
7 //const clientSecret = 'NDJiYjc50TAtMDVkMi00YzRmLTk2YjktY2EwYzU3Y2ZkZWVj';
8 // Define JWT claims
9 const payload = {
10   iss: 'https://au-syd.appid.cloud.ibm.com/management/v4/', // Replace <region> and <tenant_id> with your values
11   sub: '1234567890', // Replace with the subject for the authentication
12   aud: 'https://eu-gb.appid.cloud.ibm.com/oauth/v4/21aa5aae-c640-4524-98cd-24bc5b49dae4', // Audience (App ID region URL)
13   exp: Math.floor(Date.now() / 1000) + (60 * 5), // Token expiration time (5 minutes from now)
14   iat: Math.floor(Date.now() / 1000) // Issued at time
15 };
16 // #10-16 const payload =
17
18 // Generate the signed JWT
19 const token = jwt.sign(payload, privateKey, { algorithm: 'RS256' });
20 console.log('Generated JWT:', token);
21 /**
22 // Define the authentication URL for App ID
23 const authUrl = 'https://eu-gb.appid.cloud.ibm.com/oauth/v4/21aa5aae-c640-4524-98cd-24bc5b49dae4/token'; // Replace <region> and <tenant_id>
24
25 // Function to authenticate with App ID
26 async function authenticateWithAppID() {
27   try {
28     // Make a request to exchange the JWT for an access token
29     const response = await axios.post(authUrl, { grant_type: 'urn:ietf:params:oauth:grant-type:jwt-bearer&assertion=' + encodeURIComponent(token) },
30       headers: {
31         'Content-Type': 'application/x-www-form-urlencoded',
32         'Authorization': 'Basic ' + Buffer.from(`${clientId}:${clientSecret}`).toString('base64')
33       }
34     );
35
36     // Display the access token
37     console.log('Access Token:', response.data);
38   } catch (error) {
39     // Handle authentication error
40     console.error('Authentication error:', error.response ? error.response.data : error.message);
41   }
42 }
```

A red box highlights the line 'iss: 'https://au-syd.appid.cloud.ibm.com/management/v4/<redacted>''. To the right of this line, the text 'management url from service credentialsb' is displayed.

Name - Yash Lakhtariya

Enrollment number - 21162101012

Branch - CBA      Batch - 71

CS Practical 12

The screenshot shows the Visual Studio Code interface with the file `app.js` open. The code is used to generate a JSON Web Token (JWT) for an App ID. Key parts of the code include:

```
const jwt = require('jsonwebtoken');
const fs = require('fs');
const axios = require('axios');
const privateKey = fs.readFileSync('private_key.pem', 'utf-8'); // Replace with the actual path to your private key file
const clientId = 'd098e898-494c-4b6d-b0d5-68664732d54a';
const clientSecret = 'NDJ1Yjcs0TATMDVkmI00YzRmLTh2YjktY2EwYzU3Y2ZkZWVj';
const payload = {
  iss: 'https://au-syd.appid.cloud.ibm.com/management/v4',
  sub: 'Yashlani access token maate', // Replace with the subject for the authentication
  aud: 'https://au-syd.appid.cloud.ibm.com/oauth/v4/E', // Audience (App ID region URL)
  exp: Math.floor(Date.now() / 1000) + (60 * 5), // Token expiration time (5 minutes from now)
  iat: Math.floor(Date.now() / 1000) // Issued at time
};
```

A red box highlights the `aud` field, which contains the URL `'https://au-syd.appid.cloud.ibm.com/oauth/v4/E'`. A tooltip for this field says "oAuthServer URL here". The status bar at the bottom indicates the file has 13 lines and 89 characters.

The screenshot shows the Visual Studio Code interface with the file `app2.js` open. This file is similar to `app.js` but uses a different URL for the audience. The code is as follows:

```
const jwt = require('jsonwebtoken');
const fs = require('fs');
// const axios = require('axios');
// Load your private key
const privateKey = fs.readFileSync('private_key.pem', 'utf-8'); // Replace with the actual path to your private key file
//const clientId = 'd098e898-494c-4b6d-b0d5-68664732d54a';
//const clientSecret = 'NDJ1Yjcs0TATMDVkmI00YzRmLTh2YjktY2EwYzU3Y2ZkZWVj';
const payload = {
  iss: 'https://au-syd.appid.cloud.ibm.com/management/v4/87b71da3-5cc0-b003-7cdc47f4a3bd', // Replace <region> and <tenant_id> with
  sub: 'Yashlani access token maate', // Replace with the subject for the authentication
  aud: 'https://au-syd.appid.cloud.ibm.com/oauth/v4/87b71da3-5cc0-b003-7cdc47f4a3bd', // Audience (App ID region URL)
  exp: Math.floor(Date.now() / 1000) + (60 * 20), // Token expiration time (20 minutes from now)
  iat: Math.floor(Date.now() / 1000) // Issued at time
};
```

The status bar at the bottom indicates the file has 15 lines and 89 characters. The terminal pane shows the command `node "/home/ysl/Documents/sem7practicals/CS/Practical_12/app2.js"` was run.

Name - Yash Lakhtariya  
Enrollment number - 21162101012  
Branch - CBA      Batch - 71  
CS Practical 12

The screenshot shows the IBM Cloud App ID Dashboard. On the left, there's a sidebar with various options like Overview, Manage Authentication, Cloud Directory, Identity Providers, SAML 2.0 Federation, Facebook, Google, IBMID, and Custom Identity. Under Custom Identity, there are sub-options: Profiles and roles, Login Customization, Applications, Service credentials, and Plan. A 'Test' button is highlighted with a red box. The main content area is titled 'Custom Identity' and contains a note about using your own custom identity provider. It shows a 'Public Key' section with a placeholder for the key content. Below that, another note explains how App ID uses a public key to validate JSON Web Tokens (JWT). The URL in the browser is <https://cloud.ibm.com/appid?panelid=customIdentity&tenantId=crn%3Av1%3Abluemix%3Apublic%3Aappid%3Aau-syd%3Aa%2F9553f5f7184dd922a056f240c778ef6%3A87b71da3-5cc47f4a3bd%3A%3AII/customide...>.

The screenshot shows the 'App ID Custom Identity Test' page. At the top, it says 'Test Custom Identity Flow'. It provides instructions for constructing a signed JSON Web Token (JWT) and exchanging it for an App ID token. Below this is a large text input field labeled 'JSON Web Token (JWT)' where the JWT is pasted. A 'Test' button is located at the bottom right of the input field. The URL in the browser is [https://au-syd.appid.cloud.ibm.com/custom\\_identity/v1/87b71da3-5cc47f4a3bd/test\\_app?code=Pdp5wq3DgZCP8O9wphFwq3DmyINWmgkP1YUFTDl1wOw45FdXPCgXQsLbCucOCwoMcb8K9ekbCsmvCqkoUwpHChMO...](https://au-syd.appid.cloud.ibm.com/custom_identity/v1/87b71da3-5cc47f4a3bd/test_app?code=Pdp5wq3DgZCP8O9wphFwq3DmyINWmgkP1YUFTDl1wOw45FdXPCgXQsLbCucOCwoMcb8K9ekbCsmvCqkoUwpHChMO...).

Name - Yash Lakhtariya

Enrollment number - 21162101012

Branch - CBA      Batch - 71

CS Practical 12

The screenshot shows a browser window with the title "Custom Identity Test - Vivaldi". The address bar displays a URL starting with "https://au-syd.appid.cloud.ibm.com/custom\_identity/v1/87b71da3-5ccd-4dc0-b003-7cdc47f4a3bd/test\_app?code=PDp5wq3DgZYCP8O9wphFwq3DmyINWmgkP1YuTFTD1wOw45FdXPCgXQsLbCucOCwoMcb8K9ekbCsmvCqkoJwpHChMO...". The main content area contains two sections: "Decoded Access Token" and "Decoded Identity Token", each showing JSON representations of the respective tokens.

**Decoded Access Token**

```
{
  "iss": "https://auth-appid",
  "exp": 1729747731,
  "aud": [
    "-----"
  ],
  "sub": "0-----",
  "amr": [
    "appid_custom"
  ],
  "iat": 1729744131,
  "tenant": "-----ibd",
  "scope": "openid appid_default appid_readuserattr appid_readprofile appid_writeuserattr appid_authenticated"
}
```

**Decoded Identity Token**

```
{
  "iss": "https://auth-appid",
  "exp": 1729747731,
  "aud": [
    "-----"
  ],
  "tenant": "-----",
  "iat": 1729744131,
  "sub": "-----",
  "identities": [
    {
      "provider": "appid_custom",
      "id": "Yashlani access token maate"
    }
  ]
}
```

Name - Yash Lakhtariya  
Enrollment number - 21162101012  
Branch - CBA      Batch - 71  
CS Practical 12

6. To access public keys using API, authentication is required, for which one time process for getting IAM access token is required

The screenshot shows the IBM Cloud Identity & Access Management (IAM) interface in a browser. The URL is <https://cloud.ibm.com/iam/apikeys>. The left sidebar has a tree view with 'IBM Cloud' selected, then 'IAM', followed by 'Overview', 'Dashboard', 'Manage identities' (which is expanded), 'Users', 'Trusted profiles', 'Service IDs' (which is expanded), and 'API keys' (which is highlighted with a red box). Below this is 'Identity providers', 'Manage access', 'Access groups', 'Authorizations', 'Roles', 'Gain insight', and 'Settings'. Under 'Documentation', there are links to 'IBM Cloud IAM Docs' and 'Enterprise IAM Docs'. The main content area is titled 'API keys' and contains a message: 'Create, view, and work with API keys that you have access to manage. IBM Cloud API keys are associated with a user's identity and can be used to access cloud platform and classic infrastructure APIs, depending on the access that is assigned to the user. The following table displays a list of API keys created in this account.' It also says 'Looking for more options to manage API Keys? Try IBM Cloud® Secrets Manager for creating and leasing API keys dynamically and storing them securely in your own dedicated instance.' A search bar says 'View: My IBM Cloud API keys'. A table header includes columns for 'Status', 'Name', 'Description', and 'Date created'. A 'Create' button is at the top right of the table area. Below the table, it says 'No API keys' and 'Looks like there aren't any API keys. Click Create to get started.'

Name - Yash Lakhtariya  
Enrollment number - 21162101012  
Branch - CBA      Batch - 71  
CS Practical 12

The screenshot shows the 'API keys' section of the IBM Cloud IAM interface. A modal window titled 'Create IBM Cloud API key' is open. In the 'Name' field, 'yashlani-api-key-iam' is entered. The 'Leaked action' section has 'Disable the leaked key' selected. The 'Session creation' section has 'No' selected. At the bottom right of the modal is a blue 'Create' button.

The screenshot shows the 'API keys' section of the IBM Cloud IAM interface. A green success message box is displayed, stating 'Create API key' and 'The API key was successfully created.' Below the message, there is a modal window titled 'API key successfully created' showing a redacted API key value. At the bottom of this modal are 'Copy' and 'Download' buttons, with the 'Download' button being highlighted.

Name - Yash Lakhtariya  
Enrollment number - 21162101012  
Branch - CBA      Batch - 71  
CS Practical 12

7. Use nodejs code to get IAM Access token using API key we generated

Code :

```
const axios = require('axios');

async function getIamToken(apiKey) {
  const url = 'https://iam.cloud.ibm.com/identity/token';

  try {
    const response = await axios.post(url, null, {
      headers: {
        'Content-Type': 'application/x-www-form-urlencoded',
      },
      params: {
        apikey: apiKey,
        grant_type: 'urn:ibm:params:oauth:grant-type:apikey',
      },
    });
    return response.data.access_token; // Return the access token
  } catch (error) {
    console.error('Error obtaining IAM token:', error.response.data);
```

Name - Yash Lakhtariya  
Enrollment number - 21162101012  
Branch - CBA      Batch - 71  
CS Practical 12

```
        throw error;
    }
}

// Use your API key
const apiKey = 'xxxxxxxxxxxxxxxxxxxx';
getIamToken(apiKey)
    .then(token => {
        console.log('IAM Access Token:', token);
        // Now you can use this token in your API requests
    })
    .catch(err => {
        console.error('Failed to get IAM token:', err);
});
```

Name - Yash Lakhtariya  
Enrollment number - 21162101012  
Branch - CBA      Batch - 71  
CS Practical 12

code-url-handler test2.js - Practical-12 - Visual Studio Code

```
steps to perform.docx app.js test2.js
1 const axios = require('axios');
2
3 async function getIAMToken(apiKey) {
4   const url = 'https://iam.cloud.ibm.com/identity/token';
5
6   try {
7     const response = await axios.post(url, null, {
8       headers: {
9         'Content-Type': 'application/x-www-form-urlencoded',
10      },
11      params: {
12        apiKey: apiKey,
13        grant_type: 'urn:ibm:params:oauth:grant-type:apikey',
14      },
15    }); <- #7-15 const response = await axios.post
16    return response.data.access_token; // Return the access token
17  } catch (error) {
18    console.error('Error obtaining IAM token:', error.response.data);
19    throw error;
20  }
21 } <- #3-21 async function getIAMToken(apiKey)
22
23 // Use your API key
24 const apiKey = 'REDACTED'
25 getIAMToken(apiKey)
26 .then(token => {
27   console.log('IAM Access Token:', token);
28   // Now you can use this token in your API requests
29 })
30 .catch(err => {
31   console.error('Failed to get IAM token:', err);
32 });
33
```

paste api key here

code-url-handler test2.js - Practical-12 - Visual Studio Code

```
File Edit Selection View Go Run Terminal Help
steps to perform.docx app.js test2.js
Ln 26, Col 21 Spaces: 4 UTF-8 CRLF () JavaScript Go Live ✓ Prettier
```

File Edit Selection View Go Run Terminal Help
steps to perform.docx app.js test2.js
Ln 15, Col 12 Spaces: 4 UTF-8 CRLF () JavaScript Go Live ✓ Prettier

```
EXPLORER PRACTICAL-12 node_modules app.js know.json package-lock.json package.json steps to perform.docx test.js test2.js
test2.js > f, then() callback
1 const axios = require('axios');
2
3 async function getIAMToken(apiKey) {
4   const url = 'https://iam.cloud.ibm.com/identity/token';
5
6   try {
7     const response = await axios.post(url, null, {
8       headers: {
9         'Content-Type': 'application/x-www-form-urlencoded',
10      },
11      params: {
12        apiKey: apiKey,
13        grant_type: 'urn:ibm:params:oauth:grant-type:apikey',
14      },
15    }); <- #7-15 const response = await axios.post
16    return response.data.access_token; // Return the access token
17  } catch (error) {
18    console.error('Error obtaining IAM token:', error.response.data);
19    throw error;
20  }
21 } <- #3-21 async function getIAMToken(apiKey)
22
23 // Use your API key
24 const apiKey = 'REDACTED'
25 getIAMToken(apiKey)
26 .then(token => {
27   console.log('IAM Access Token:', token);
28   // Now you can use this token in your API requests
29 })
30 .catch(err => {
31   console.error('Failed to get IAM token:', err);
32 });
33
```

TERMINAL

```
node "/home/ysl/Downloads/Practical-12/test2.js"
  0s ... ↵ / ↵ /Practical-12
  • node "/home/ysl/Downloads/Practical-12/test2.js"
  IAM Access Token:
  REDACTED
  2s ... ↵ / ↵ /Practical-12
  •
```

Name - Yash Lakhtariya  
Enrollment number - 21162101012  
Branch - CBA      Batch - 71  
CS Practical 12

When API is used to get public key from IBM Cloud App ID using JWT and then verifying it, authentication is required otherwise anyone can access our public key. Hence for this purpose, this IAM Access Token will be used

8. Now, use nodejs another code to test access to resources using this IAM access token got as output

Code :

```
const express = require('express');
const jwt = require('jsonwebtoken');
const axios = require('axios');

const app = express();
const port = 3000;

// Your App ID public key URL for verifying the JWT signature
const APP_ID_PUBLIC_KEY_URL =
'https://au-syd.appid.cloud.ibm.com/management/v4/xxxxxxxxxxxxxx
xxxxxxxxxx/config/idps/custom';

// Middleware to verify the JWT
async function verifyToken(req, res, next) {
  const authHeader = req.headers['authorization'];

  if (!authHeader || !authHeader.startsWith('Bearer ')) {
    return res.status(401).json({ error: 'Authorization' })
```

Name - Yash Lakhtariya  
Enrollment number - 21162101012  
Branch - CBA      Batch - 71  
CS Practical 12

```
header missing or malformed' });
}

const token = authHeader.split(' ')[1];

try {
    // Fetch the App ID public keys
    const response = await axios.get(APP_ID_PUBLIC_KEY_URL, {
        headers: {
            Authorization: `Bearer xxxxxxxxxxxxxxxx` // Pass the
        Bearer token here
        },
    });
    console.log(response)
    const publicKeys = response.data.config.publicKey;
    console.log(publicKeys);
    if (!publicKeys || publicKeys.length === 0) {
        return res.status(500).json({ error: 'Failed to
retrieve public keys' });
    }

    // Decode the JWT header to find the key ID (kid)
    const decodedHeader = jwt.decode(token, { complete: true
});
```

Name - Yash Lakhtariya  
Enrollment number - 21162101012  
Branch - CBA      Batch - 71  
CS Practical 12

```
if (!decodedHeader || !decodedHeader.header) {
    return res.status(400).json({ error: 'Invalid token
format' });
}

// Find the corresponding public key based on the kid
//const key = publicKeys.find(k => k.kid ===
decodedHeader.header.kid);

if (!publicKeys) {
    return res.status(400).json({ error: 'Public key not
found for token' });
}

// Construct the PEM formatted public key
// const publicKey = `-----BEGIN PUBLIC
KEY-----\n${publicKeys}\n-----END PUBLIC KEY-----`;
//console.log(publicKey);
// Verify the token using the public key
jwt.verify(token, publicKeys, { algorithms: ['RS256'] },
(err, decoded) => {
    if (err) {
        return res.status(401).json({ error: 'Token
verification failed', message: err.message });
    }
});
```

Name - Yash Lakhtariya  
Enrollment number - 21162101012  
Branch - CBA      Batch - 71  
CS Practical 12

```
}

    // Token is valid, attach user info to request and
proceed

        req.user = decoded;
        next();
    });

} catch (error) {
    res.status(500).json({ error: 'Failed to verify token',
message: error.message });
}

// Sample protected route
app.get('/protected', verifyToken, (req, res) => {
    res.json({ message: 'Access granted to protected resource',
user: req.user });
});

// Start the server
app.listen(port, () => {
    console.log(`Server running at http://localhost:${port}`);
});
```

Name - Yash Lakhtariya

Enrollment number - 21162101012

Branch - CBA      Batch - 71

CS Practical 12

A screenshot of the Visual Studio Code interface. The title bar shows "code-urhandler test.js - Practical\_12 - Visual Studio Code". The status bar at the bottom right indicates "Ln 20, Col 1 Spaces: 4 UTF-8 LF () JavaScript ⌘ Go Live ⌘ Prettier". The code editor displays a file named "test.js" with the following content:

```
const express = require('express');
const jwt = require('jsonwebtoken');
const axios = require('axios');

const app = express();
const port = 3000;

// Your App ID public key URL for verifying the JWT signature
const APP_ID_PUBLIC_KEY_URL = 'https://au-syd.appid.cloud.ibm.com/management/v4/config/idps/custom';

// Middleware to verify the JWT
async function verifyToken(req, res, next) {
  const authHeader = req.headers['authorization'];

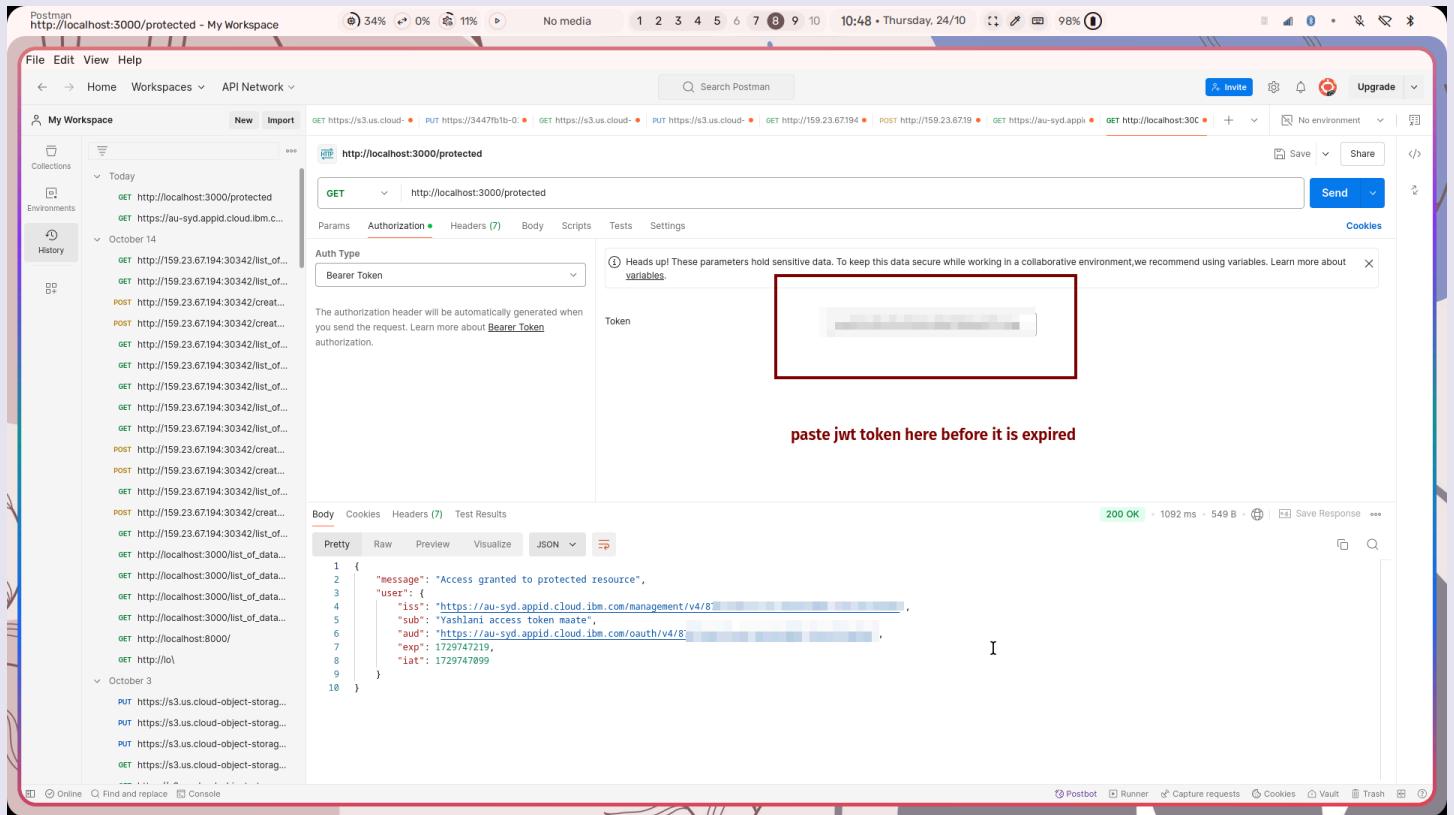
  if (!authHeader || !authHeader.startsWith('Bearer ')) {
    return res.status(401).json({ error: 'Authorization header missing or malformed' });
  }

  const token = authHeader.split(' ')[1];
  try {
    // Fetch the App ID public keys
    const response = await axios.get(APP_ID_PUBLIC_KEY_URL, {
      headers: {
        Authorization: `Bearer ${token}`
      }
    });
    const publicKeys = response.data.config.publicKey;
    console.log(publicKeys);
    if (!publicKeys || publicKeys.length === 0) {
      return res.status(500).json({ error: 'Failed to retrieve public keys' });
    }

    // Decode the JWT header to find the key ID (kid)
    const decodedHeader = jwt.decode(token, { complete: true });
    if (!decodedHeader || !decodedHeader.header) {
      return res.status(400).json({ error: 'Invalid token format' });
    }
  }
}
```

The line "paste iam access token here to authorize for jwt" is highlighted with a red box. The "Authorization" header in the axios.get call is also highlighted with a red box.

**Name - Yash Lakhtariya**  
**Enrollment number - 21162101012**  
**Branch - CBA      Batch - 71**  
**CS Practical 12**

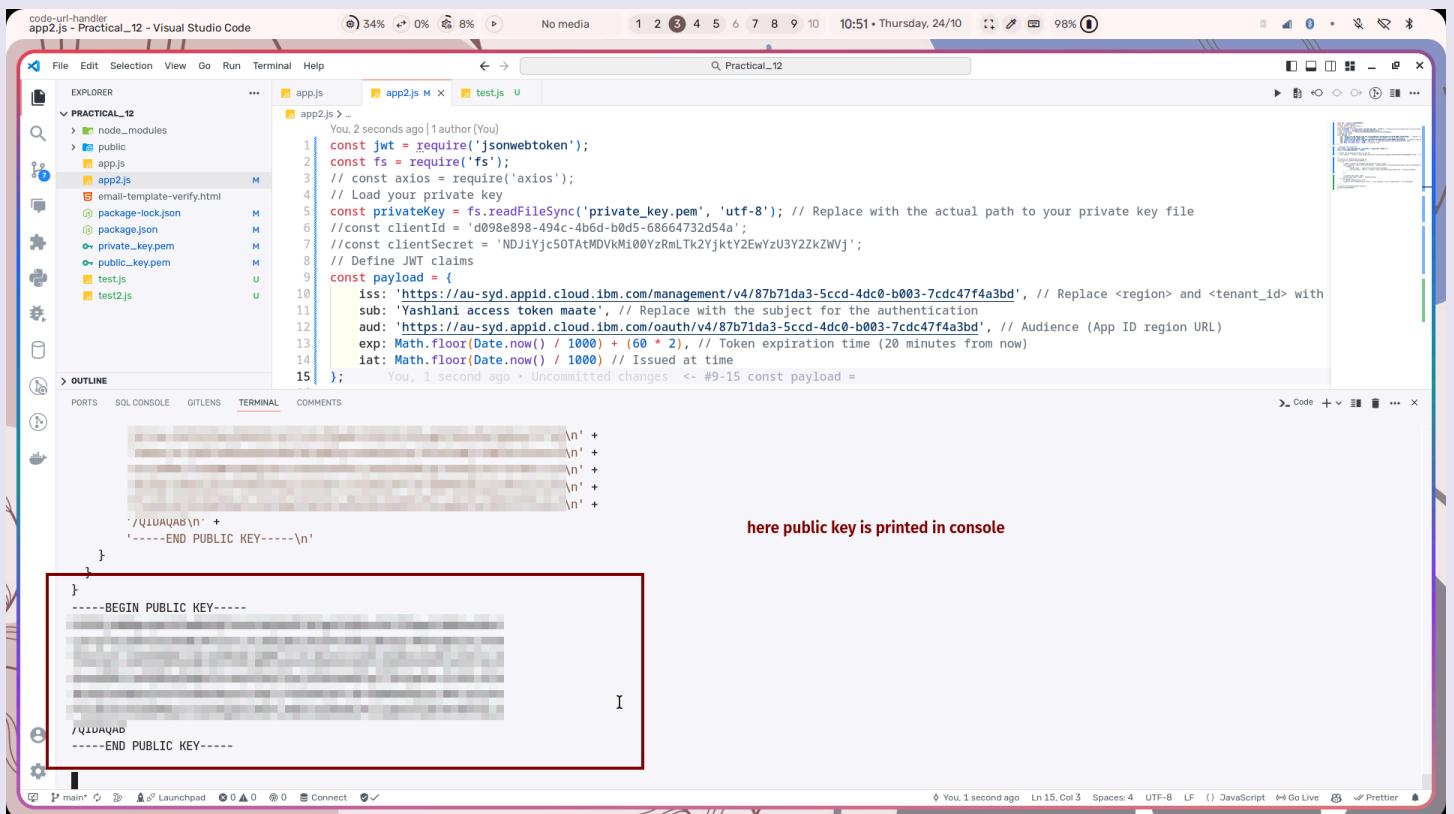


The screenshot shows the Postman application interface. The URL in the address bar is `http://localhost:3000/protected`. The request method is set to `GET`, and the endpoint is `http://localhost:3000/protected`. The `Authorization` tab is selected, showing `Bearer Token` as the auth type. A red box highlights the token input field, which contains a placeholder text: `paste jwt token here before it is expired`. The response status is `200 OK` with a response time of `1092 ms` and a size of `549 B`. The response body is displayed in JSON format:

```
1 {
2   "message": "Access granted to protected resource",
3   "user": {
4     "iss": "https://au-syd.appid.cloud.ibm.com/management/v4/8",
5     "sub": "Yashlani access token maate",
6     "aud": "https://au-syd.appid.cloud.ibm.com/oauth/v4/8",
7     "exp": 1729747219,
8     "iat": 1729747099
9   }
10 }
```

In this code, the public key in App ID is verified using JWT as authentication but IAM access token is also used for the purpose of accessing public key in authorized manner from cloud

**Name - Yash Lakhtariya**  
**Enrollment number - 21162101012**  
**Branch - CBA      Batch - 71**  
**CS Practical 12**



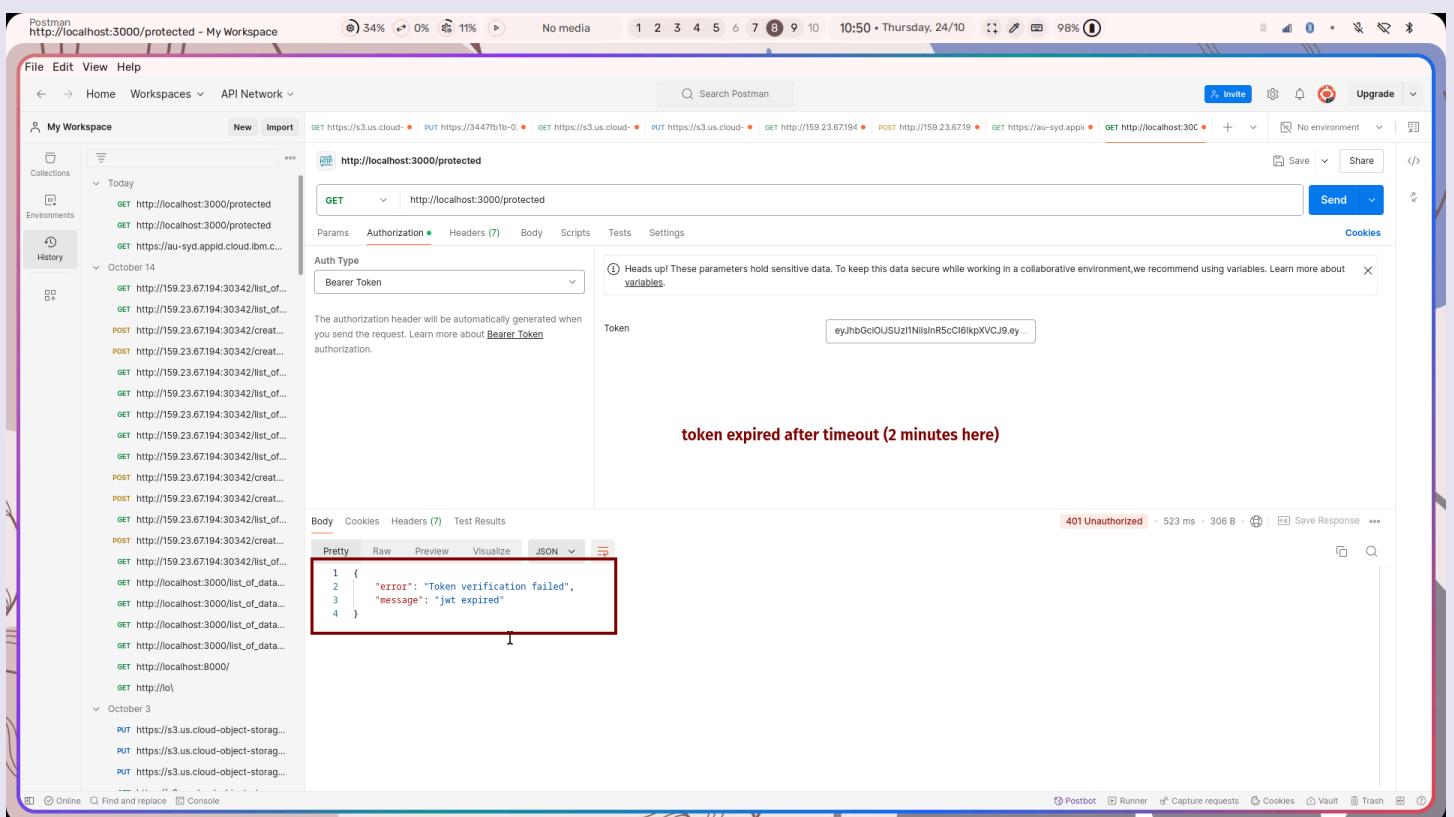
The screenshot shows the Visual Studio Code interface with the file `app2.js` open. The code generates a JWT payload and prints it to the console. A red box highlights the public key portion of the output.

```
const jwt = require('jsonwebtoken');
const fs = require('fs');
// const axios = require('axios');
// Load your private key
const privateKey = fs.readFileSync('private_key.pem', 'utf-8'); // Replace with the actual path to your private key file
//const clientId = 'd098e898-494c-4bbd-b0d5-68664732d54a';
//const clientSecret = 'NDJ1Yjcs0TAtMDVhM100yZmLTh2YjktY2EwYzU3Y2ZkZWVj';
// Define JWT claims
const payload = {
  iss: 'https://au-syd.appid.cloud.ibm.com/management/v4/87b71da3-5ccd-4dc0-b003-7cdc47f4a3bd', // Replace <region> and <tenant_id> with
  sub: 'Yashlani access token maate', // Replace with the subject for the authentication
  aud: 'https://au-syd.appid.cloud.ibm.com/oauth/v4/87b71da3-5ccd-4dc0-b003-7cdc47f4a3bd', // Audience (App ID region URL)
  exp: Math.floor(Date.now() / 1000) + (60 * 2), // Token expiration time (20 minutes from now)
  iat: Math.floor(Date.now() / 1000) // Issued at time
};

You, 1 second ago • Uncommitted changes <- #9-15 const payload =
```

here public key is printed in console

```
-----BEGIN PUBLIC KEY-----  
-----END PUBLIC KEY-----  
-----BEGIN PUBLIC KEY-----  
-----END PUBLIC KEY-----  
/Q1URQAB  
-----END PUBLIC KEY-----
```



The screenshot shows the Postman application interface. A red box highlights the error message in the response body, which indicates a token verification failed because it has expired.

token expired after timeout (2 minutes here)

```
1  {
2    "error": "Token verification failed",
3    "message": "jwt expired"
4 }
```

Name - Yash Lakhtariya  
Enrollment number - 21162101012  
Branch - CBA      Batch - 71  
CS Practical 12

## Conclusion :

