200 XP

Exercise - Different types of tokens used in Microsoft identity

15 minutes

In this exercise, you'll create an Azure AD application and single page application for a user to sign in and display their information on the page.

Create a Node.js web application

Note

The instructions below assume you are using v2.14.2 of the Microsoft Authentication Library for JavaScript 2.0.

Open your command prompt, navigate to a directory where you want to save your work, create a new folder, and change directory into that folder.

Execute the following command to create a new Node.js application:

shell

```shell
npm init -y
```

Install the Node.js webserver **express** and HTTP request middleware **morgan** into the application:

shell

```shell
npm install express morgan
```

Open the application in Visual Studio Code using the following command:

Console

```
code .
```

Create a new file **server.js** in the root of the folder and add the following JavaScript to it. This code will start the web server:

JavaScript Copy

```javascript
var express = require('express');
var app = express();
var morgan = require('morgan');
var path = require('path');

var port = 3007;
app.use(morgan('dev'));

// set the front-end folder to serve public assets.
app.use(express.static('web'));

// set up our one route to the index.html file.
app.get('*', function (req, res) {
  res.sendFile(path.join(__dirname + '/index.html'));
});

// Start the server.
app.listen(port);
console.log(`Listening on port ${port}...`);
console.log('Press CTRL+C to stop the web server...');
```

Create a web page for the user to sign in and display details

Create a new folder **web** in the current folder and add a new file **index.html** to the folder. Add the following code to the **index.html** file:

HTML Copy

```html
<!DOCTYPE html>
<html>
<head>
  <title>Getting Started with Microsoft identity</title>
  <script
src="https://cdnjs.cloudflare.com/ajax/libs/bluebird/3.7.2/bluebird.min.js"></script>
  <script src="https://alcdn.msauth.net/browser/2.11.1/js/msal-browser.js"></script>
</head>
```

```
<body>
  <div class="container">
    <div>
      <p id="WelcomeMessage">Microsoft Authentication Library For Javascript
(MSAL.js) Exercise</p>
      <button id="SignIn" onclick="signIn()">Sign In</button>
    </div>
    <div>
      <pre id="json"></pre>
    </div>
  </div>
  <script>
    var ua = window.navigator.userAgent;
    var msie = ua.indexOf('MSIE ');
    var msie11 = ua.indexOf('Trident/');
    var msedge = ua.indexOf('Edge/');
    var isIE = msie > 0 || msie11 > 0;
    var isEdge = msedge > 0;

    var msalConfig = {
      auth: {
        clientId: '',
        authority: '',
        redirectURI: ''
      },
      cache: {
        cacheLocation: "localStorage",
        storeAuthStateInCookie: isIE || isEdge
      }
    };

    var graphConfig = {
      graphMeEndpoint: "https://graph.microsoft.com/v1.0/me",
      requestObj: {
        scopes: ["user.read"]
      }
    };

    var msalApplication = new msal.PublicClientApplication(msalConfig);
    var userName = "";
    var loginType = isIE ? "REDIRECT" : "POPUP";


    // TODO: add CODE before this line



    // TODO: add FUNCTIONS before this line
  </script>
```

```
</body>
</html>
```

Note

The remainder of this exercise instructs you to add code to this **index.html** file. Pay close attention where you add the code using the using the two `TODO:` comments for placement.

Add the following function to the **index.html** file immediately before the `// TODO: add FUNCTIONS before this line` comment that will configure the welcome message for the page:

JavaScript Copy

```
function updateUserInterface() {
  var divWelcome = document.getElementById('WelcomeMessage');
  divWelcome.innerHTML = 'Welcome <strong>' + userName + '</strong> to Microsoft
Graph API';

  var loginbutton = document.getElementById('SignIn');
  loginbutton.innerHTML = 'Sign Out';
  loginbutton.setAttribute('onclick', 'signOut();');
}
```

Next, add the following function to **index.html** immediately before the `// TODO: add FUNCTIONS before this line` comment. This function requests an access token from Microsoft identity and submits a request to Microsoft Graph for the current user's information. The function uses the popup approach for modern browsers and it uses the redirect approach for Internet Explorer:

JavaScript Copy

```
function acquireTokenAndGetUser() {
  var request = graphConfig.requestObj;
  request.account = msalApplication.getAccountByUsername(userName);

  msalApplication.acquireTokenSilent(request)
    .then(function (tokenResponse) {
      getUserFromMSGraph(tokenResponse.accessToken, graphAPICallback);
    })
    .catch(function (error) {
      console.log("silent token acquisition fails.");
      if (error instanceof msal.InteractionRequiredAuthError) {
        if (loginType == "POPUP") {
          msalApplication.acquireTokenPopup(request)
            .then(function (tokenResponse) {
              getUserFromMSGraph(tokenResponse.accessToken, graphAPICallback);
            })
            .catch(function (error) { console.error(error); }
            );
        } else {
          msalApplication.acquireTokenRedirect(request);
        }
      } else {
        console.error(error);
      }
    });
}
```

The function first attempts to retrieve the access token silently from the currently signed in user. If the user needs to sign in, the function will trigger either the popup or redirect authentication process.

The redirect approach to authenticating requires an extra step. The MSAL application on the page needs to see if the current page was requested based on a redirect from Azure AD. If so, it needs to process information in the URL request provided by Azure AD.

Add the following code immediately before the `// TODO: add CODE before this line` comment:

JavaScript Copy

```
msalApplication.handleRedirectPromise()
  .then(handleResponse)
  .catch(function (error) { console.log(error); }
  );
```

Once the user is authenticated, the code can submit a request to Microsoft Graph for the current user's information. The `acquireTokenAndGetUser()` function passes the access token acquired from Azure AD to the `getUserFromMSGraph()` function you are about to add.

Add the following functions immediately before the `// TODO: add FUNCTIONS before this line` comment:

JavaScript Copy

```
function getUserFromMSGraph(accessToken, callback) {
  var endpoint = graphConfig.graphMeEndpoint;

  var xmlHttp = new XMLHttpRequest();
  xmlHttp.onreadystatechange = function () {
    if (this.readyState == 4 && this.status == 200)
```

```
      callback(JSON.parse(this.responseText));
    }
    xmlHttp.open("GET", endpoint, true);
    xmlHttp.setRequestHeader('Authorization', 'Bearer ' + accessToken);
    xmlHttp.send();
  }

  function graphAPICallback(data) {
    document.getElementById("json").innerHTML = JSON.stringify(data, null, 2);
  }
```

Finally, add the following functions to implement a sign in and sign out capability for the button on the page.

Add the functions immediately before the `// TODO: add FUNCTIONS before this line` comment:

JavaScript Copy

```javascript
function handleResponse(loginResponse) {
  if (loginResponse != null) {
    userName = loginResponse.account.username;
  } else {
    var currentAccounts = msalApplication.getAllAccounts();
    if (currentAccounts == null || currentAccounts.length == 0) {
      return;
    } else {
      userName = currentAccounts[0].username;
    }
  }

  updateUserInterface();
  acquireTokenAndGetUser();
}

function signIn() {
  if (loginType == "POPUP") {
    msalApplication.loginPopup(graphConfig.requestObj)
      .then(handleResponse)
      .catch(function (error) { console.log(error); }
      );
  } else {
    msalApplication.loginRedirect(graphConfig.requestObj);
  }
}

function signOut() {
  var logoutRequest = {
    account: msalApplication.getAccountByUsername(userName)
  };
```

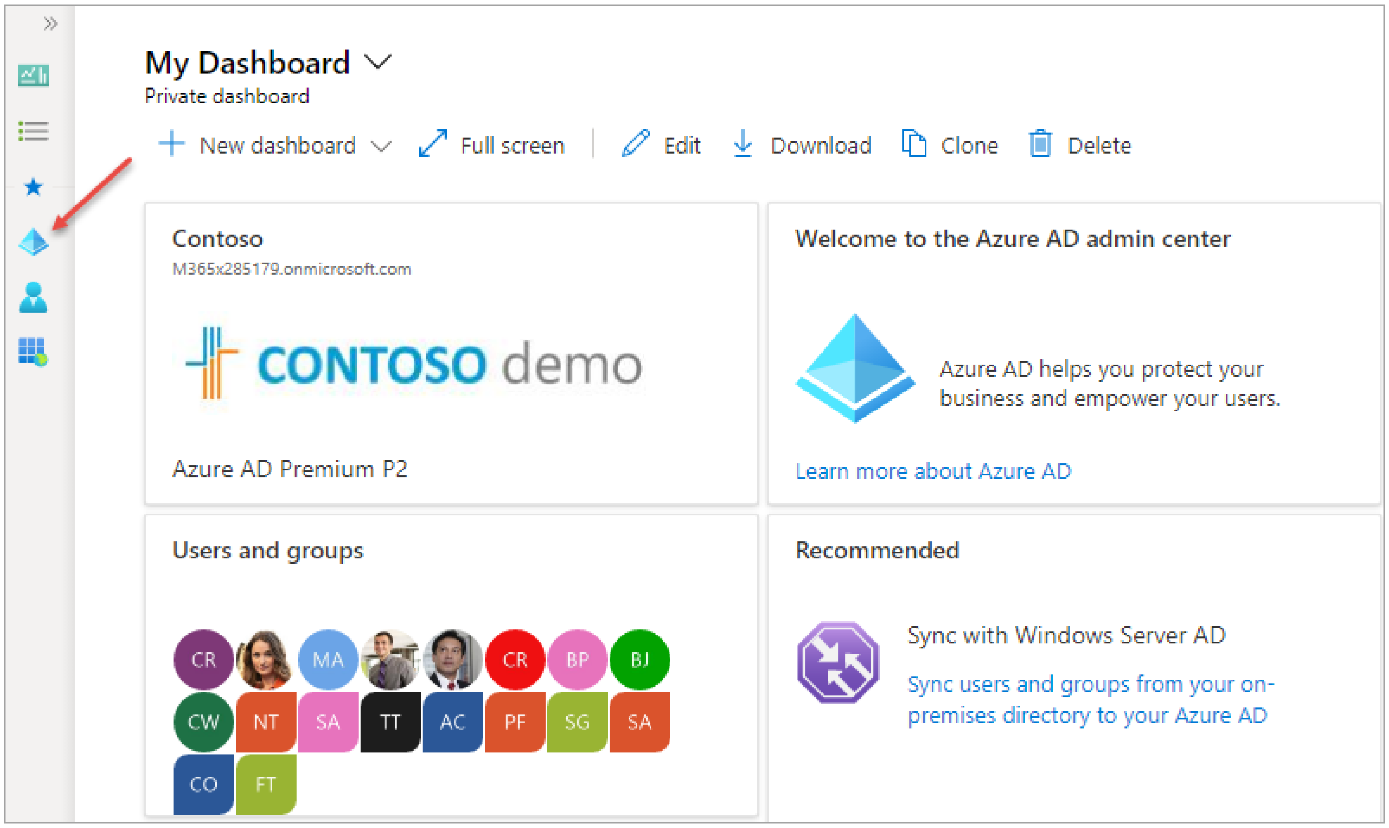
```
    msalApplication.logout(logoutRequest);
  }
```

Create an Azure AD application

The web page you created will submit a request to Microsoft Graph to retrieve the user's details. All requests to Microsoft Graph must include an access token as proof of the user's identity and that they have the necessary permissions to call Microsoft Graph. To obtain an access token, you must create an Azure AD application.

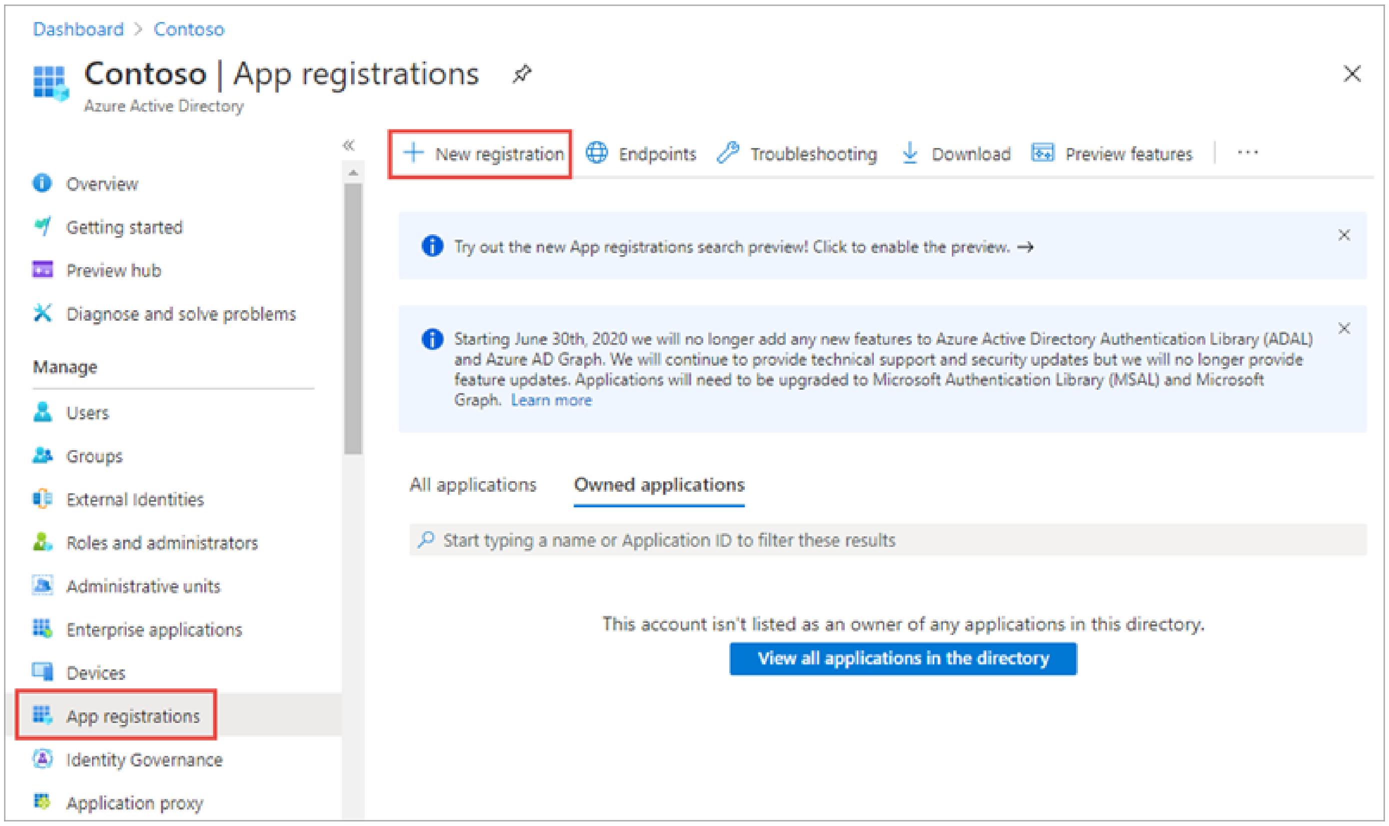
Open a browser and navigate to the Azure Active Directory admin center (https://aad.portal.azure.com) . Sign in using a **Work or School Account** that has global administrator rights to the tenancy.

Select **Azure Active Directory** in the left-hand navigation.


My Dashboard
Private dashboard
New dashboard
Full screen
Edit
Download
Clone
Delete
Contoso
M365x285179.onmicrosoft.com
CONTOSO demo
Azure AD Premium P2
Welcome to the Azure AD admin center
Azure AD helps you protect your business and empower your users.
Learn more about Azure AD
Users and groups
CR MA CR BP BJ
CW NT SA TT AC PF SG SA
CO FT
Recommended
Sync with Windows Server AD
Sync users and groups from your on-premises directory to your Azure AD

Select **Manage** > **App registrations** in the left-hand navigation.

On the **App registrations** page, select **New registration**.



Try out the new App registrations search preview! Click to enable the preview. →

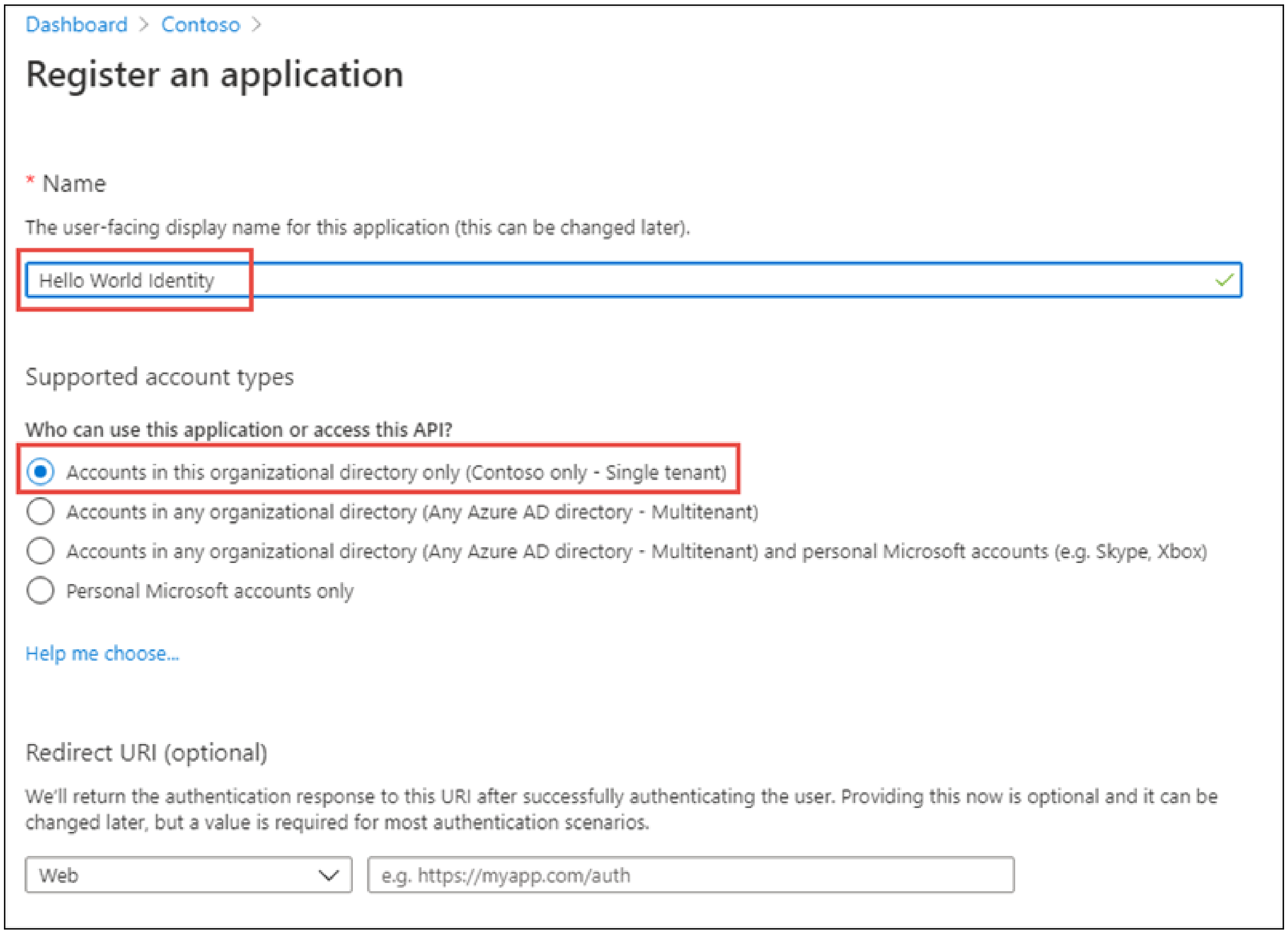
Starting June 30th, 2020 we will no longer add any new features to Azure Active Directory Authentication Library (ADAL) and Azure AD Graph. We will continue to provide technical support and security updates but we will no longer provide feature updates. Applications will need to be upgraded to Microsoft Authentication Library (MSAL) and Microsoft Graph. Learn more

Start typing a name or Application ID to filter these results

This account isn't listed as an owner of any applications in this directory.

On the **Register an application** page, set the values as follows:

- **Name**: Hello World Identity
- **Supported account types**: Accounts in this organizational directory only (Single tenant)



Dashboard > Contoso >

Register an application

* Name

The user-facing display name for this application (this can be changed later).

Hello World Identity

Supported account types

Who can use this application or access this API?

[✓ Accounts in this organizational directory only (Contoso only - Single tenant)]

Accounts in any organizational directory (Any Azure AD directory - Multitenant)

Accounts in any organizational directory (Any Azure AD directory - Multitenant) and personal Microsoft accounts (e.g. Skype, Xbox)

Personal Microsoft accounts only

Help me choose...

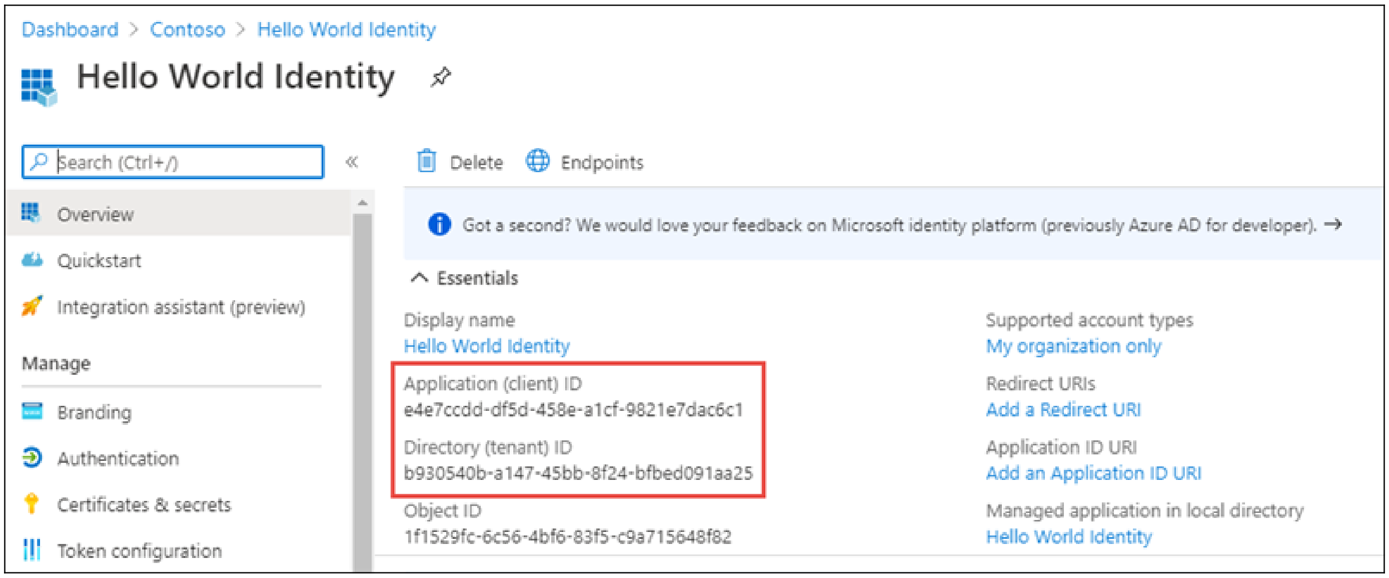
Redirect URI (optional)

We'll return the authentication response to this URI after successfully authenticating the user. Providing this now is optional and it can be changed later, but a value is required for most authentication scenarios.

Web | e.g. https://myapp.com/auth

Select **Register** to create the application.

On the **Hello World Identity** page, copy the values **Application (client) ID** and **Directory (tenant) ID**; you'll need these values later in this exercise.



Got a second? We would love your feedback on Microsoft identity platform (previously Azure AD for developer). →

Essentials

Display name: Hello World Identity

Application (client) ID: e4e7ccdd-df5d-458e-a1cf-9821e7dac6c1

Directory (tenant) ID: b930540b-a147-45bb-8f24-bfbed091aa25

Object ID: 1f1529fc-6c56-4bf6-83f5-c9a715648f82

Supported account types: My organization only

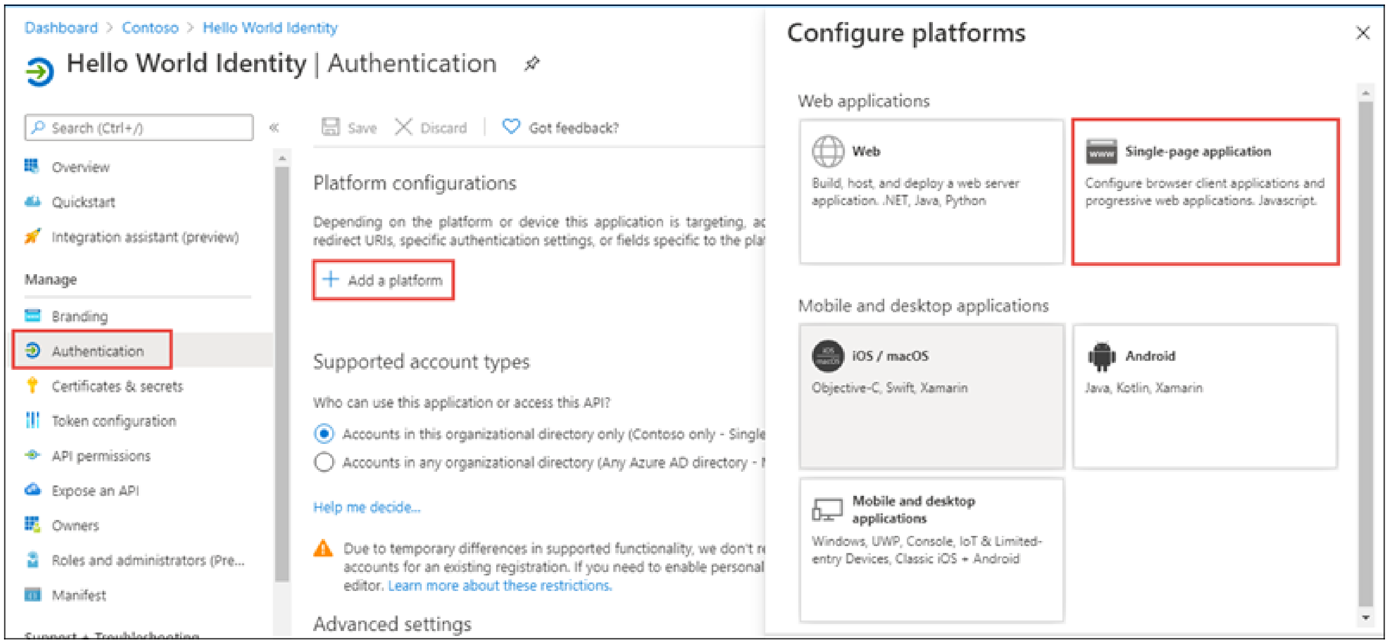
Redirect URIs: Add a Redirect URI

Application ID URI: Add an Application ID URI

Managed application in local directory: Hello World Identity

Select **Manage** > **Authentication** in the left-hand navigation.

On the **Authentication** page, select **Add a platform**. When the **Configure platforms** panel appears, select **Single-page application**.



Depending on the platform or device this application is targeting, ad
redirect URIs, specific authentication settings, or fields specific to the pla

Who can use this application or access this API?

Accounts in this organizational directory only (Contoso only - Single

Accounts in any organizational directory (Any Azure AD directory - M

Due to temporary differences in supported functionality, we don't re
accounts for an existing registration. If you need to enable personal
editor. Learn more about these restrictions.

Web: Build, host, and deploy a web server application. .NET, Java, Python

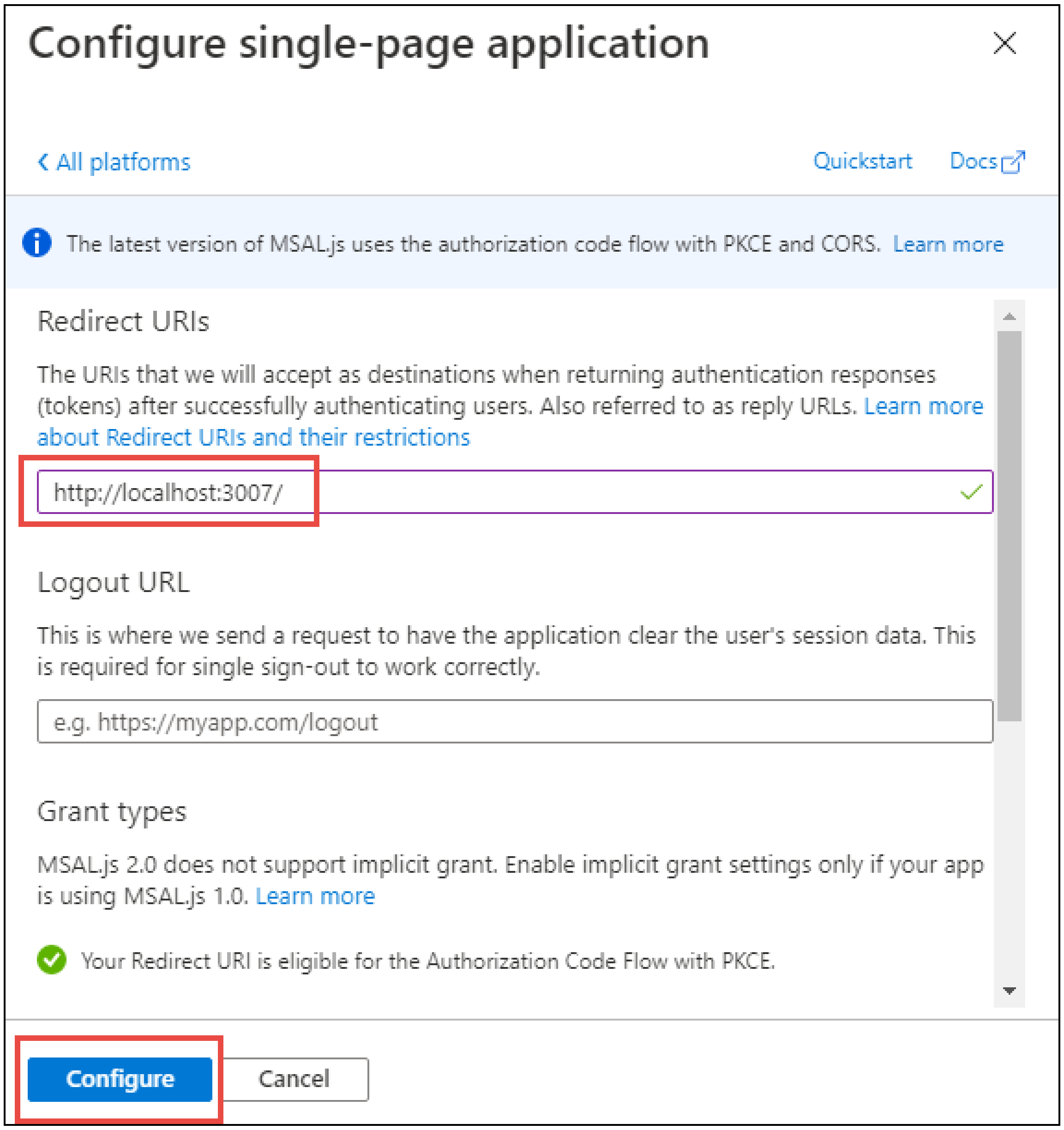
Single-page application: Configure browser client applications and progressive web applications. Javascript.

iOS / macOS: Objective-C, Swift, Xamarin

Android: Java, Kotlin, Xamarin

Mobile and desktop applications: Windows, UWP, Console, IoT & Limited-entry Devices, Classic iOS + Android

In the **Configure single-page application** panel, add **http://localhost:3007** under **Redirect URIs**, and select **Configure**.


Configure single-page application

< All platforms Quickstart Docs

The latest version of MSAL.js uses the authorization code flow with PKCE and CORS. Learn more

Redirect URIs

The URIs that we will accept as destinations when returning authentication responses (tokens) after successfully authenticating users. Also referred to as reply URLs. Learn more about Redirect URIs and their restrictions

http://localhost:3007/

Logout URL

This is where we send a request to have the application clear the user's session data. This is required for single sign-out to work correctly.

e.g. https://myapp.com/logout

Grant types

MSAL.js 2.0 does not support implicit grant. Enable implicit grant settings only if your app is using MSAL.js 1.0. Learn more

Your Redirect URI is eligible for the Authorization Code Flow with PKCE.

Update the web page with the Azure AD application details

The last step is to configure the web page to use the Azure AD application.

Locate the `var msalConfig = {}` code in the **index.html** file. The `auth` object contains three properties you need to set as follows:

- `clientId`: set to the Azure AD application's ID
- `authority`: set to **`https://login.microsoftonline.com/{{DIRECTORY_ID}}`**, replacing the {{DIRECTORY_ID}} with the Azure AD directory ID of the Azure AD application
- `redirectURI`: set to the Azure AD application's redirect URI: **http://localhost:3007**

Test the web application

Important

If you are using Internet Explorer, ensure that `http://localhost` and `https://login.microsoftonline.com` are both in the same **security zone** - **Trusted Sites** is recommended.

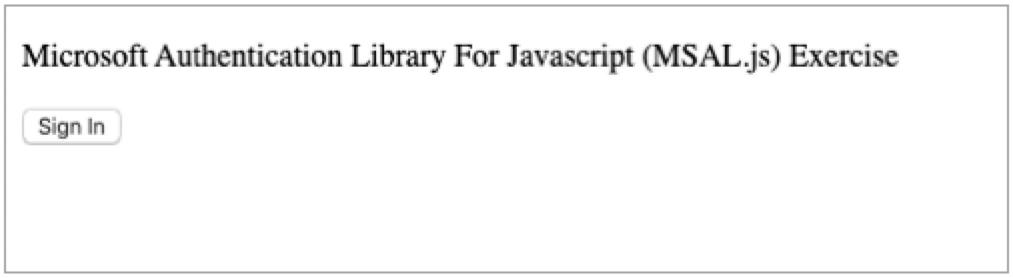
To test the web page, first start the local web server. In the command prompt, execute the following command from the root of the project:

shell

Copy

```shell
node server.js
```

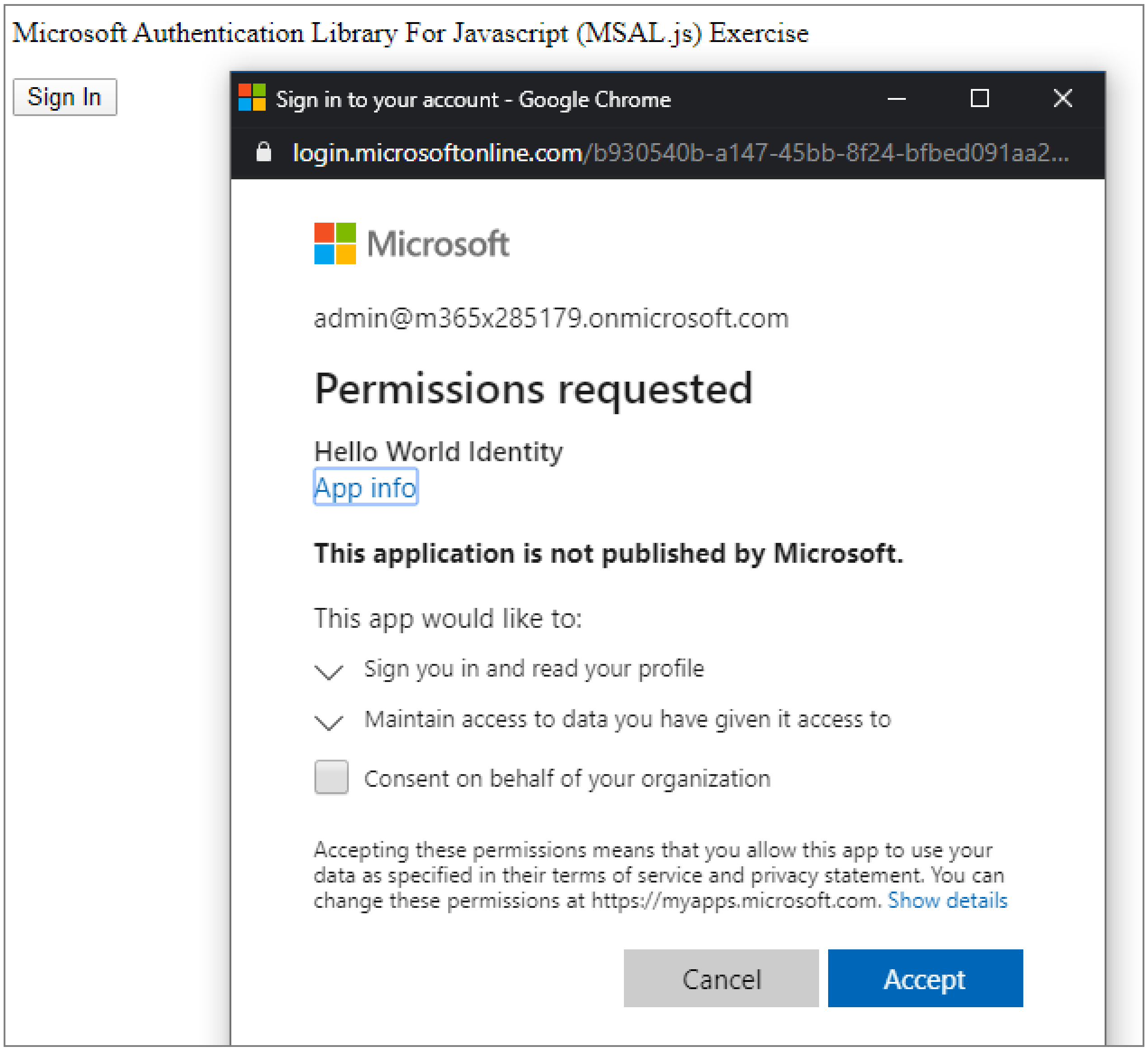
Next, open a browser and navigate to http://localhost:3007. The page initially contains a default welcome message and sign in button.


Microsoft Authentication Library For Javascript (MSAL.js) Exercise

Select the **sign in** button.

Depending on the browser, you are using, a popup window will load or the page will redirect to the Azure AD sign in prompt.

Sign in using a **Work or School Account** and accept the permissions requested for the application by selecting **Accept**


Microsoft Authentication Library For Javascript (MSAL.js) Exercise

Sign In

Sign in to your account - Google Chrome

login.microsoftonline.com/b930540b-a147-45bb-8f24-bfbed091aa2...

Microsoft

admin@m365x285179.onmicrosoft.com

Permissions requested

Hello World Identity
App info

This application is not published by Microsoft.

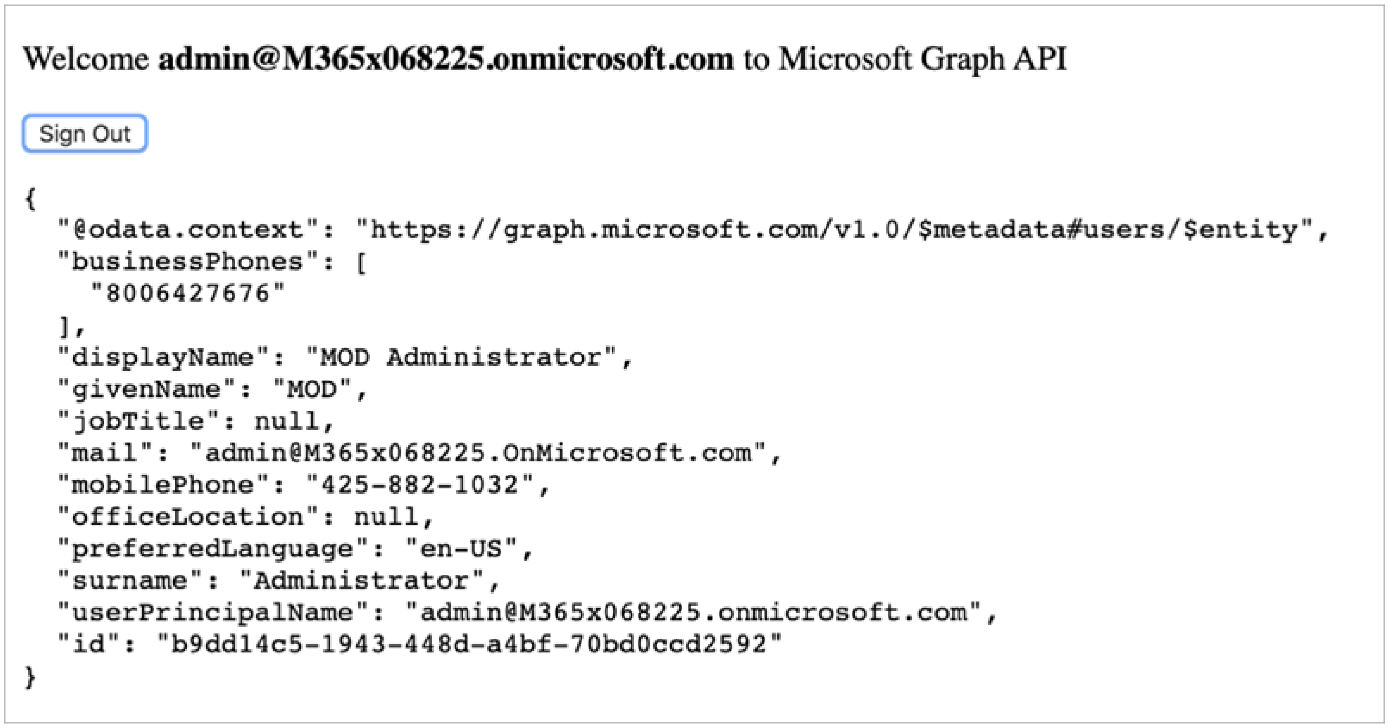
This app would like to:

- Sign you in and read your profile
- Maintain access to data you have given it access to
- Consent on behalf of your organization

Accepting these permissions means that you allow this app to use your data as specified in their terms of service and privacy statement. You can change these permissions at https://myapps.microsoft.com. Show details

Cancel Accept

Depending on the browser you're using, the popup will disappear or you will be redirected back to the web page. When the page loads, MSAL will request an access token and request your information from Microsoft Graph. After the request complete, it will display the results on the page:


Welcome **admin@M365x068225.onmicrosoft.com** to Microsoft Graph API

```
{
  "@odata.context": "https://graph.microsoft.com/v1.0/$metadata#users/$entity",
  "businessPhones": [
    "8006427676"
  ],
  "displayName": "MOD Administrator",
  "givenName": "MOD",
  "jobTitle": null,
  "mail": "admin@M365x068225.OnMicrosoft.com",
  "mobilePhone": "425-882-1032",
  "officeLocation": null,
  "preferredLanguage": "en-US",
  "surname": "Administrator",
  "userPrincipalName": "admin@M365x068225.onmicrosoft.com",
  "id": "b9dd14c5-1943-448d-a4bf-70bd0ccd2592"
}
```

Stop the local web server by pressing CTRL + C in the console.

Summary

In this exercise, you created an Azure AD application and single page application for a user to sign in and display their information on the page.

Test your knowledge

1. Which of the following statements about ID tokens is correct?

- [✓ ID tokens contain basic identity information about the currently logged in user.]

  ✓ **Correct, ID tokens are provided upon request when a user signs in. They contain basic identity information about the user, saving the application from having to issue another request for this information.**

- ID tokens can be submitted in an authentication request to prove the identity of the user.
- ID tokens include both identity information and the permissions a user has been granted to an application.

2. Which of the following statements about access tokens is incorrect?

- ○ Access tokens can be created by Azure AD either for a user or for an application.
- ○ Azure AD supports multiple OAuth 2.0 flows that developers can use to obtain an access token.
- ● Access tokens are submitted by the application to the identity provider to request permissions to a resource on behalf of a user.

✓ **This answer is incorrect. Access tokens aren't submitted by an application, they're provided by the identity platform.**

Next unit: Account types in Microsoft identity

Continue >