Web Application Development Diploma in IT/FI Year 2 (2021/22) Semester 3	Week 15
	2 hours
Practical 13 – Web API (OAuth 2.0)	

OBJECTIVES

In this practical, you will:

- Learn how to use configure a Single-Sign-On using OAuth 2.0 to a Web API server.

REFERENCES

- Google Developers - Google Identity OpenID Connect (https://developers.google.com/api-client-library/dotnet/guide/aaa_oauth)

INSTRUCTIONS

Part 1 – OAuth 2.0

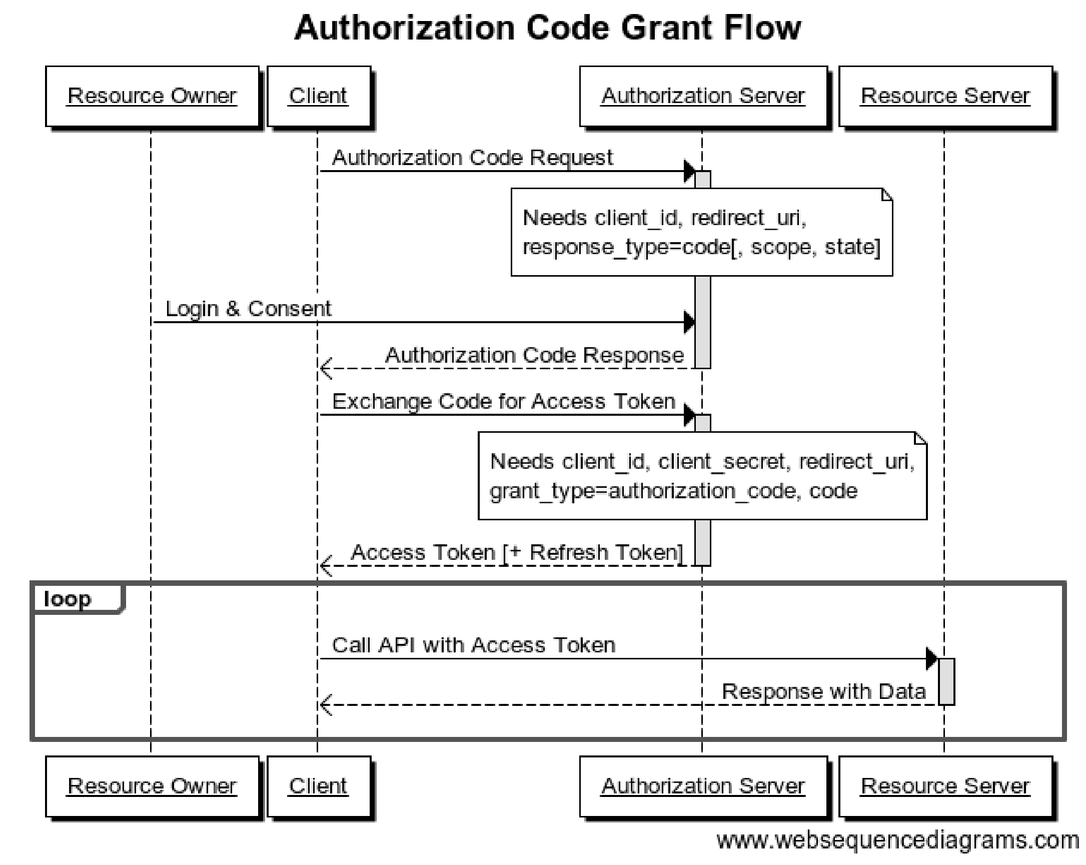
Recall from the last 2 practical exercises, you have learnt that you are able to retrieve and send JSON from the Web API server. The Web API server also allows students to vote on the popular books to purchase. Our NP Book Rental application want to know which student is carry out these activities, hence an authentication and authorization process is required. Instead of storing all students' information (including login password) in the application, we are going to make use of a third-party trust-worthy server to help us to authenticate and authorize each user. The authorization method we are using is known as OAuth2.0 and the third-party authorization server that our application is to connect to through Web API is Google.

Understanding OAuth 2.0

OAuth 2.0 is the industry-standard protocol for authorization. OAuth 2.0 focuses on **client developer** simplicity while providing specific authorization flows for **web applications**, desktop applications, mobile phones, and living room devices.

OAuth 2.0 enables applications to obtain limited access to user accounts on an HTTP service, such as Microsoft, Google, Facebook and Twitter. It works by delegating user authentication to the service that hosts the user account, and authorizing third-party applications to access the user account.

Access token is the most important part in OAuth 2.0, Access token is a piece of random string generated by the authorization server and is issued when the client requests them after being authenticated successfully. Application (client) uses access token to make API requests on behalf of a user. The access token represents the authorization of a specific application to access specific parts of a user's data. It has a limited lifetime, which is defined by the authorization server.


Authorization Code Grant Flow
Resource Owner
Client
Authorization Server
Resource Server
Authorization Code Request
Needs client_id, redirect_uri, response_type=code[, scope, state]
Login & Consent
Authorization Code Response
Exchange Code for Access Token
Needs client_id, client_secret, redirect_uri, grant_type=authorization_code, code
Access Token [+ Refresh Token]
loop
Call API with Access Token
Response with Data
www.websequencediagrams.com

OAuth 2.0 defines 4 roles:

Resource Owner: A user who owns the login credentials and personal information (e.g. Microsoft account owner, Google account owner, Facebook account owner.) The resource owner provides consent to authorization server whether the account can be used for login purpose for an application and the scope of the authorization granted (e.g. what information to be released to the application).

Resource Server: server hosting protected data (for example Google hosting individual profile and personal information).

Client: Application requesting access to a resource server (e.g. NP BookStore Web Application).

Authorization Server: Server issuing access token to the client. This token will be used for the client to request the resource server.

Scenario:
1. A website wants to obtain information about your Google profile.
2. You are redirected by the client (the website) to the authorization server (Google).
3. If you authorize access, the authorization server sends an authorization code to the client (the website) in the callback response.
4. Then, this code is exchanged against an access token between the client and the authorization server.
5. The website is now able to use this access token to query the resource server (Google again) and retrieve your profile data.

You never see the access token, it will be stored by the website (in session or cookie for example). Google also sends other information with the access token, such as the token lifetime and eventually a refresh token.

This is the ideal scenario and the safer one because the access token is not passed on the client side (web browser in our example).

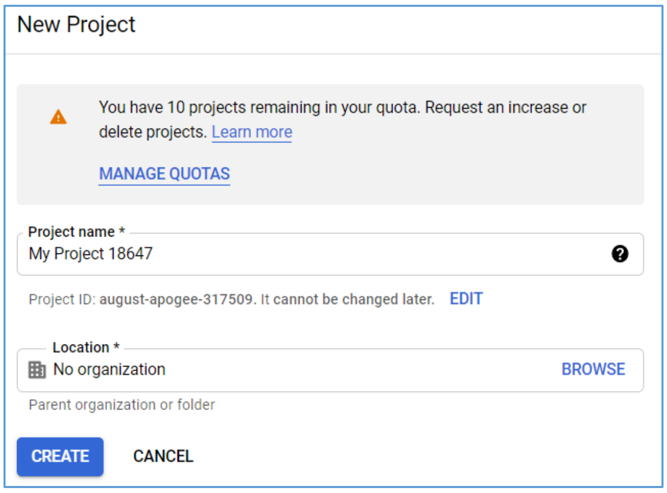
References:
https://oauth.net/2/
http://www.bubblecode.net/en/2016/01/22/understanding-oauth2/

Part 2 – Create a Google API Console Project and Client ID

Before using the Google authentication and authorization server, we need to register our application, i.e., NP Book Rental, with Google. A Client ID and Client Secret will be provided after registration, which our application will need to provide when accessing the Google authorization server. The Client ID and Client Secret helps the login provider (i.e. Google) to understand which application is requesting for the authenticated user's profile to provide user security. Google will also present the user which a consent screen for agreeing to exchange user's data with the requesting application which in this case is our web application along with information about what our application would receive from the user's profile stored in the provider.

1. Log into https://console.developers.google.com with your **personal** Google account.

2. At the top of the page, next to the **Google Cloud Platform** logo, there should be a project dropdown list. Click on that and select New Project. Give the project a name and click on the Create button.


New Project

You have 10 projects remaining in your quota. Request an increase or delete projects. Learn more

MANAGE QUOTAS

Project name *
My Project 18647

Project ID: august-apogee-317509. It cannot be changed later. EDIT

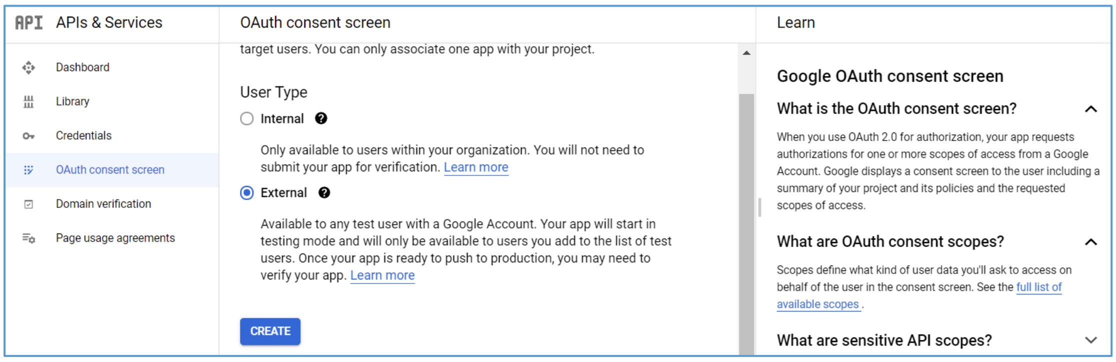
Location *
No organization BROWSE

Parent organization or folder

CREATE CANCEL

After the project is created successfully, it will appear in the project dropdown list and a dashboard for the project appears.

3. Click on the menu icon ≡ in the top-left hand side of the page. Navigate to **API & Services** and click on **OAuth consent screen.** Select the "**External**" User Type and click on the Create button.



target users. You can only associate one app with your project.

User Type

○ Internal

Only available to users within your organization. You will not need to submit your app for verification. Learn more

● External

Available to any test user with a Google Account. Your app will start in testing mode and will only be available to users you add to the list of test users. Once your app is ready to push to production, you may need to verify your app. Learn more

Google OAuth consent screen

What is the OAuth consent screen?

When you use OAuth 2.0 for authorization, your app requests authorizations for one or more scopes of access from a Google Account. Google displays a consent screen to the user including a summary of your project and its policies and the requested scopes of access.

What are OAuth consent scopes?

Scopes define what kind of user data you'll ask to access on behalf of the user in the consent screen. See the full list of available scopes .

What are sensitive API scopes?

4. On the next page, provide the following information:
   - App Name (e.g., NP Book Rental)
   - User Support Email (e.g., your G-mail account)
   - Application home page link (e.g. https://localhost:<PortNumber>/)
   - Application privacy policy (e.g. https://localhost:<PortNumber>/Home/Privacy)
   - Developer contact information (e.g., your G-mail account)

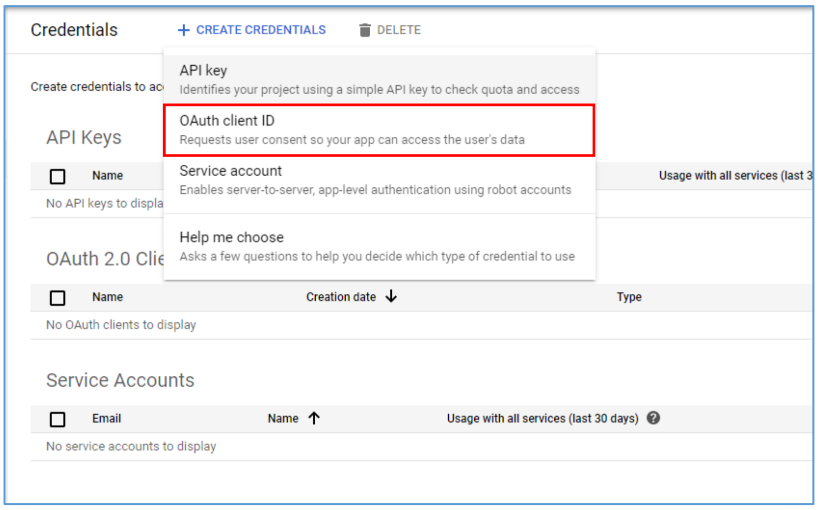
   Click on the "**Save and Continue**" button to proceed.

5. The next page is to define the scope of access of the user account. Click the "**Add or Remove Scopes**" button to add the following scopes:
   - auth/userinfo.email
   - auth/userinfo.profile

   Remember to click the "**Update**" button after selecting the above scopes from a checkbox list. Click on the "**Save and Continue**" button to proceed.

6. Skip the "Test Users" page. Click on the "**Save and Continue**" button to proceed. After this page, you will see a summary of all the information you have entered. You may edit it if need to.

7. With OAuth consent completed, we can now proceed to create an OAuth credential. Navigate to **API & Services** and click on **Credentials**. Click on the **Create Credentials** button and select **OAuth client ID**.



Create credentials to ac

API key
Identifies your project using a simple API key to check quota and access

OAuth client ID
Requests user consent so your app can access the user's data

Service account
Enables server-to-server, app-level authentication using robot accounts

Help me choose
Asks a few questions to help you decide which type of credential to use

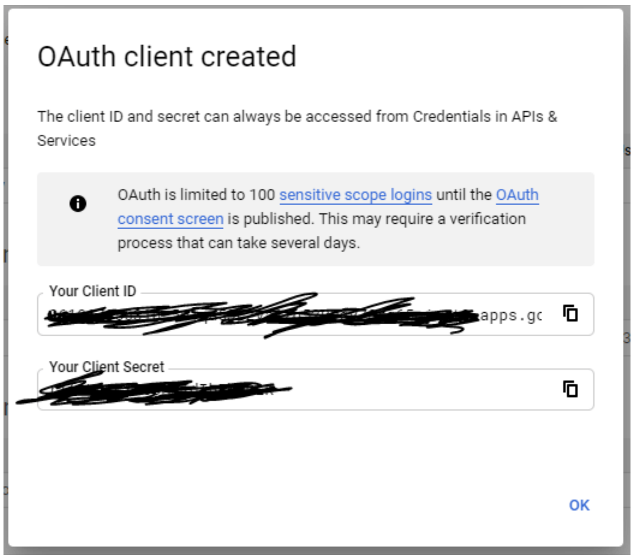
No API keys to displa

No OAuth clients to display

No service accounts to display

8. On the page that appears, enter the following pieces of information:
   - Application type: select **Web application**
   - Name: e.g., NP Book Rental
   - Authorized redirect URL: e.g. https://localhost:<PortNumber>/signin-oidc

   The "signin-oidc" route is handled by the ASP.NET Core OpenID connect authentication workflow middleware for Google, which we will install and configure in the next section. Once done, click on the **Create** button and you have a popup showing up the Client ID and Client Secret.



OAuth client created

The client ID and secret can always be accessed from Credentials in APIs & Services

OAuth is limited to 100 sensitive scope logins until the OAuth consent screen is published. This may require a verification process that can take several days.

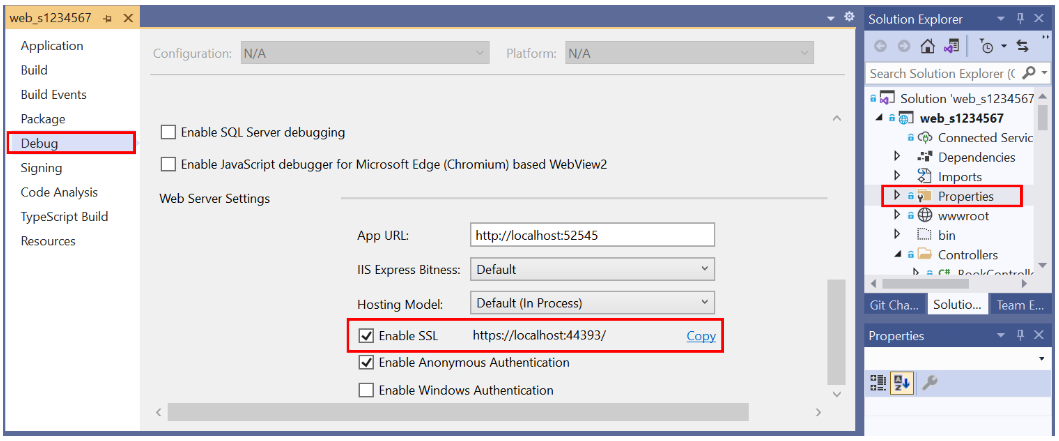
Your Client ID
[redacted]apps.gc

Your Client Secret
[redacted]

Copy done the Client ID and Client Secret on Notepad as you will need to use them in our ASP.NET Core application.

The redirect URL is used for the Google authorization server to send the Authorization Code and Access Token. You may do the following steps to check the URL and port number used by your web application from Visual Studio:

- Double-click at the "Properties" item in Solution Explorer to open the Properties panel for your ASP.NET Core Web App project.
- Select "Debug" option from the left side of the panel, scroll down until you see the URL besides the "Enable SSL" field.



Configuration: N/A   Platform: N/A

☐ Enable SQL Server debugging
☐ Enable JavaScript debugger for Microsoft Edge (Chromium) based WebView2

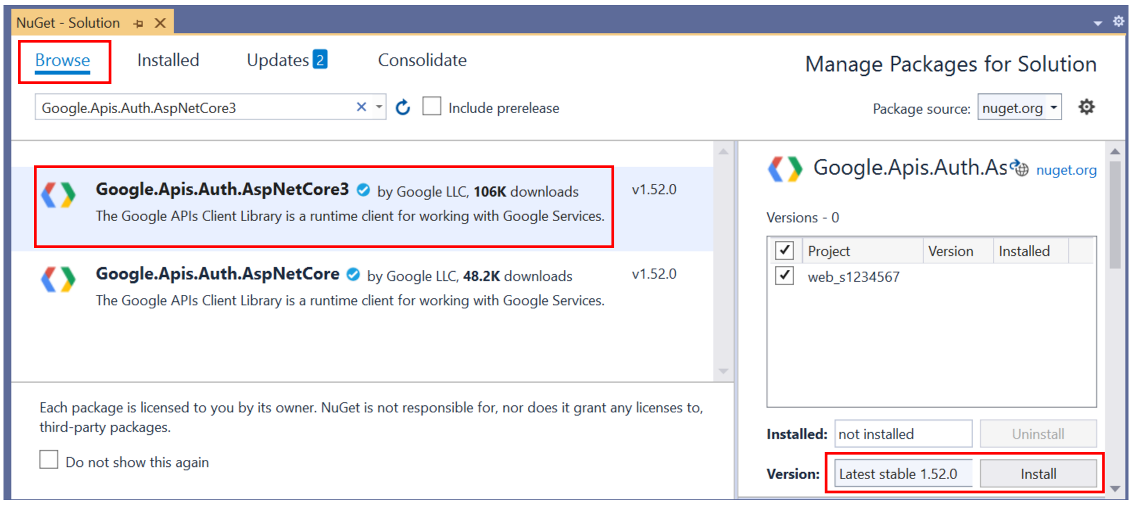
Web Server Settings

App URL: http://localhost:52545
IIS Express Bitness: Default
Hosting Model: Default (In Process)
☑ Enable SSL https://localhost:44393/ Copy
☑ Enable Anonymous Authentication
☐ Enable Windows Authentication

Part 3 – Single Sign-On (SSO) with Google

9. Launch **Visual Studio** and open the project created in the earlier practical exercise.

10. We will use a library package to help our web application to connect to the Google authentication and authorization server. At the Solution Explorer, right-click the "Solution" of your web project and select "**Manage NuGet Packages for Solution**", at the NuGet Manager, browse for the package: "**Google.Apis.Auth.AspNetCore3**" and install this library into your project.


NuGet - Solution
Browse
Installed
Updates 2
Consolidate
Manage Packages for Solution
Google.Apis.Auth.AspNetCore3
Include prerelease
Package source: nuget.org

Google.Apis.Auth.AspNetCore3 by Google LLC, **106K** downloads v1.52.0
The Google APIs Client Library is a runtime client for working with Google Services.

Google.Apis.Auth.AspNetCore by Google LLC, **48.2K** downloads v1.52.0
The Google APIs Client Library is a runtime client for working with Google Services.

Each package is licensed to you by its owner. NuGet is not responsible for, nor does it grant any licenses to, third-party packages.

Do not show this again

Google.Apis.Auth.As nuget.org

Versions - 0

✓	Project	Version	Installed
✓	web_s1234567		

Installed: not installed Uninstall

Version: Latest stable 1.52.0 Install

11. Open the open the ***appsettings.json*** file from Solution Explorer, Add the highlighted JSON entries that contains the Google Client ID and Client Secret that you have created previously:

```
{
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft": "Warning",
      "Microsoft.Hosting.Lifetime": "Information"
    }
  },
  "AllowedHosts": "*",
  "ConnectionStrings": {
    "NPBookConnectionString":
      "Data Source=(localdb)\\MSSQLLocalDB;Initial Catalog=NPBookWEB;
       Integrated Security=True;"
  },
  "Authentication": {
    "Google": {
      "ClientId": "[redacted]",
      "ClientSecret": "[redacted]"
    }
  }
}
```

12. Open the ***Startup.cs*** file from Solution Explorer, add the highlighted namespace and C# code in ***ConfigureServices()*** method to set up Google authentication:

```
using Microsoft.AspNetCore.Authentication.Cookies;
using Google.Apis.Auth.AspNetCore3;
```

```
public void ConfigureServices(IServiceCollection services)
{
    . . . . . .
    services.AddControllersWithViews();

    // This configures Google.Apis.Auth.AspNetCore3 for use in this app.
    services.AddAuthentication(options =>
    {
        // Login (Challenge) to be handled by Google OpenID Handler,
        options.DefaultChallengeScheme =
                GoogleOpenIdConnectDefaults.AuthenticationScheme;

        // Once a user is authenticated, the OAuth2 token info
        // is stored in cookies.
        options.DefaultScheme =
                CookieAuthenticationDefaults.AuthenticationScheme;
    })
    .AddCookie()
    .AddGoogleOpenIdConnect(options =>
    {
        // Credentials (stored in appsettings.json) to identify
        // the web app when performing Google authentication
        options.ClientId =
                Configuration["Authentication:Google:ClientId"];
        options.ClientSecret =
                Configuration["Authentication:Google:ClientSecret"];
    });
}
```

13. In the ***Configure()*** method, add the **authentication middleware** (highlighted code).

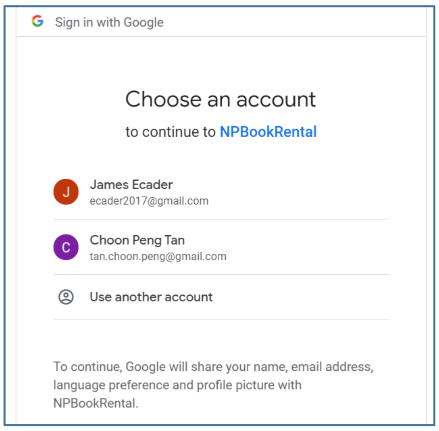
```
public void Configure(IApplicationBuilder app, IHostingEnvironment env)
{
    . . . . . .

    app.UseRouting();
    app.UseAuthentication();
    app.UseAuthorization();

    // Enable session state
    // IMPORTANT: This session call MUST go before UseMvc()
    app.UseSession();

    // Define the default route of MVC
    app.UseMvc(routes =>
    {
        routes.MapRoute(
            name: "default",
            template: "{controller=Home}/{action=Index}/{id?}");
    });
}
```

14. Add the highlighted namespaces and replace the ***StudentLogin()*** action method in **HomeController** with the highlighted shown as follow. The **[Authorize]** attribute will trigger the Google authentication that we defined in **Startup.cs**, if the authentication cookie is absent or expired. (Note: Google authentication cookie and access token will expire in 1 hour.)

```
using Microsoft.AspNetCore.Authentication.Cookies;
using Google.Apis.Auth.OAuth2;
using Microsoft.IdentityModel.Protocols.OpenIdConnect;
using Google.Apis.Auth;
using static Google.Apis.Auth.GoogleJsonWebSignature;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Authentication;

[Authorize]
public async Task<ActionResult> StudentLogin()
{
    // The user is already authenticated, so this call won't
    // trigger login, but it allows us to access token related values.
    AuthenticateResult auth = await HttpContext.AuthenticateAsync();
    string idToken = auth.Properties.GetTokenValue(
        OpenIdConnectParameterNames.IdToken);
    try {
        // Verify the current user logging in with Google server
        // if the ID is invalid, an exception is thrown
        Payload currentUser = await
                GoogleJsonWebSignature.ValidateAsync(idToken);
        string userName = currentUser.Name;
        string eMail = currentUser.Email;
        HttpContext.Session.SetString("LoginID", userName + " / "
                                        + eMail);
        HttpContext.Session.SetString("Role", "Student");
        HttpContext.Session.SetString("LoggedInTime",
                                        DateTime.Now.ToString());
        return RedirectToAction("Index", "Book");
    }
    catch (Exception e) {
        // Token ID is may be tempered with, force user to logout
        return RedirectToAction("LogOut");
    }
}
```

15. Select **File**->**Save** … (or press **<Ctrl>+<S>**) to save the controller file.

16. Test the page using Google Chrome and observe the result with **the Google user name** and email address appear on the label.
    - Click the "**Sign in with Google**" button in the Student Login panel
    - The Google Authentication page will appear, log in using your Google personal account


Sign in with Google

Choose an account

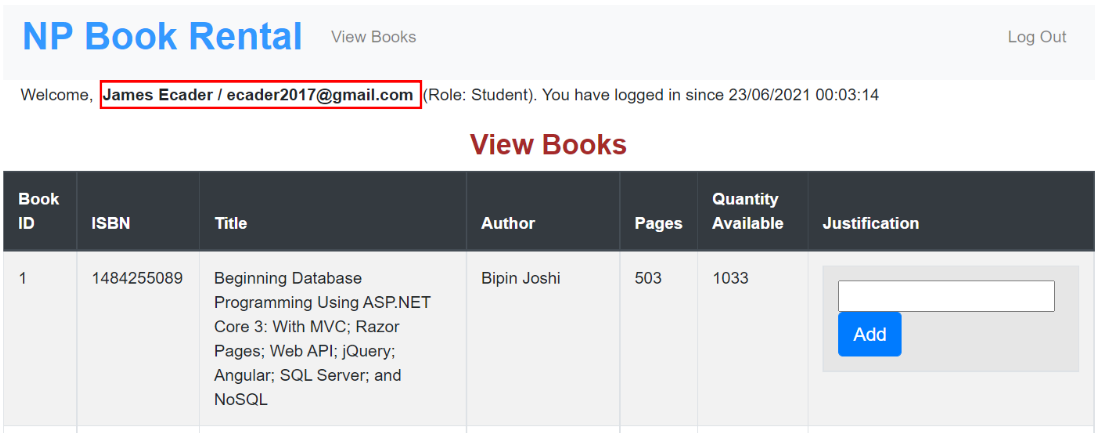
to continue to NPBookRental

James Ecader
ecader2017@gmail.com

Choon Peng Tan
tan.choon.peng@gmail.com

Use another account

To continue, Google will share your name, email address, language preference and profile picture with NPBookRental.



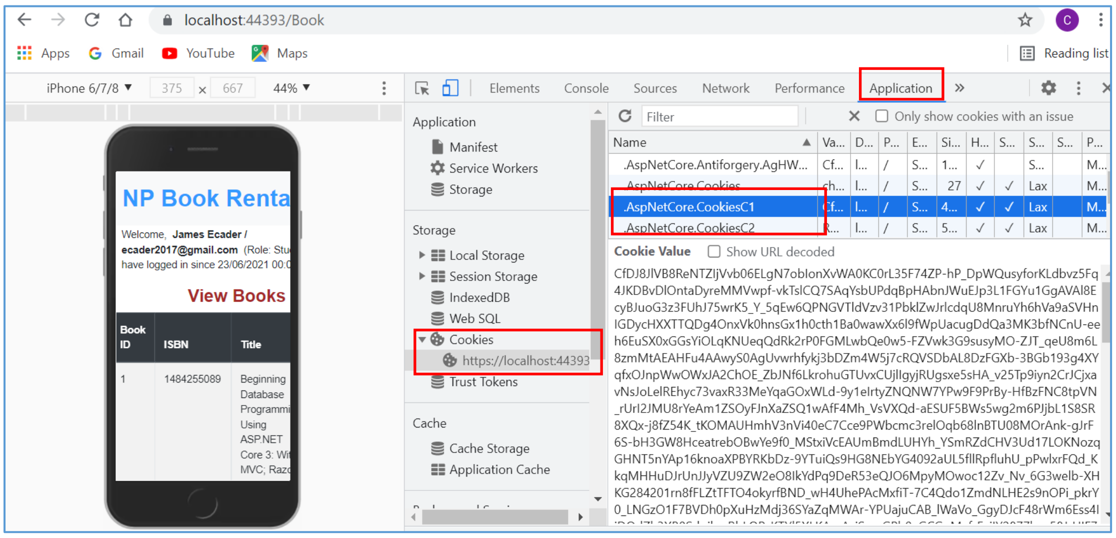
NP Book Rental View Books Log Out

Welcome, James Ecader / ecader2017@gmail.com (Role: Student). You have logged in since 23/06/2021 00:03:14

View Books

Book ID	ISBN	Title	Author	Pages	Quantity Available	Justification
1	1484255089	Beginning Database Programming Using ASP.NET Core 3: With MVC; Razor Pages; Web API; jQuery; Angular; SQL Server; and NoSQL	Bipin Joshi	503	1033	Add

17. Press function <F12> to open the Debugger of Google Chrome, Click at the "**Application**" tab and select **Storage > Cookies** field, you will see 2 cookie chunks (.AspNetCore.CokkiesC1 and .AspNetCore.CokkiesC2) created for our web app. These cookie chunks contain the access token and ID token which is created after Google authorization server granted access.



Welcome, James Ecader / ecader2017@gmail.com (Role: Stu have logged in since 23/06/2021 00:0

Name	Va...	D...	P...	E...	Si...	H...	S...	S...	S...	P...
.AspNetCore.Antiforgery.AgHW...	Cf...	l...	/	S...	1...	✓		S...		M...
.AspNetCore.Cookies	ch...	l...	/	S...	27	✓	✓	Lax		M...
.AspNetCore.CookiesC1	Cf...	l...	/	S...	4...	✓	✓	Lax		M...
.AspNetCore.CookiesC2	R...	l...	/	S...	5...	✓	✓	Lax		M...

Cookie Value Show URL decoded

CfDJ8JlVB8ReNTZIjVvb06ELgN7obIonXvWA0KC0rL35F74ZP-hP_DpWQusyforKLdbvz5Fq
4JKDBvDlOntaDyreMMVwpf-vkTslCQ7SAqYsbUPdqBpHAbnJWuEJp3L1FGYu1GgAVAl8E
cyBJuoG3z3FUhJ75wrK5_Y_5qEw6QPNGVTldVzv31PbklZwJrlcdqU8MnruYh6hVa9aSVHn
IGDycHXXTTQDg4OnxVk0hnsGx1h0cth1Ba0wawXx6l9fWpUacugDdQa3MK3bfNCnU-ee
h6EuSX0xGGsYiOLqKNUeqQdRk2rP0FGMLwbQe0w5-FZVwk3G9susyMO-ZJT_qeU8m6L
8zmMtAEAHFu4AAwyS0AgUvwrhfykj3bDZm4W5j7cRQVSDbAL8DzFGXb-3BGb193g4XY
qfxOJnpWwOWxJA2ChOE_ZbJNf6LkrohuGTUvxCUjlIgyjRUgsxe5sHA_v25Tp9iyn2CrJCjxa
vNsJoLelREhyc73vaxR33MeYqaGOxWLd-9y1eIrtyZNQNW7YPw9F9PrBy-HfBzFNC8tpVN
_rUrI2JMU8rYeAm1ZSOyFJnXaZSQ1wAfF4Mh_VsVXQd-aESUF5BWs5wg2m6PJjbL1S8SR
8XQx-j8fZ54K_tKOMAUHmhV3nVi40eC7Cce9PWbcmc3relOqb68lnBTU08MOrAnk-gJrF
6S-bH3GW8HceatrebOBwYe9f0_MStxiVcEAUmBmdLUHYh_YSmRZdCHV3Ud17LOKNozq
GHNT5nYAp16knoaXPBYRKbDz-9YTuiQs9HG8NEbYG4092aUL5fllRpfluhU_pPwlxrFQd_K
kqMHHuDJrUnJJyVZU9ZW2eO8IkYdPq9DeR53eQJO6MpyMOwoc12Zv_Nv_6G3welb-XH
KG284201rn8fFLZtTFTO4okyrfBND_wH4UhePAcMxfiT-7C4Qdo1ZmdNLHE2s9nOPi_pkrY
0_LNGzO1F7BVDh0pXuHzMdj36SYaZqMWAr-YPUajuCAB_lWaVo_GgyDJcF48rWm6Ess4l

18. Stop the testing, open the **BookController** and **VoteController** class files, add **[Authorize]** attribute before the class definition so we can restrict access to all action methods in these 2 controllers by authorized users only.

```
using Microsoft.AspNetCore.Authorization;
. . . . . . .
    [Authorize]
    public class BookController : Controller
```

```
using Microsoft.AspNetCore.Authorization;
. . . . . . .
    [Authorize]
    public class VoteController : Controller
```

19. Select **File**->**Save All** … (or press **<Ctrl>+<S>**) to save the controller files.

Part 3 – Logout from Single Sign-On

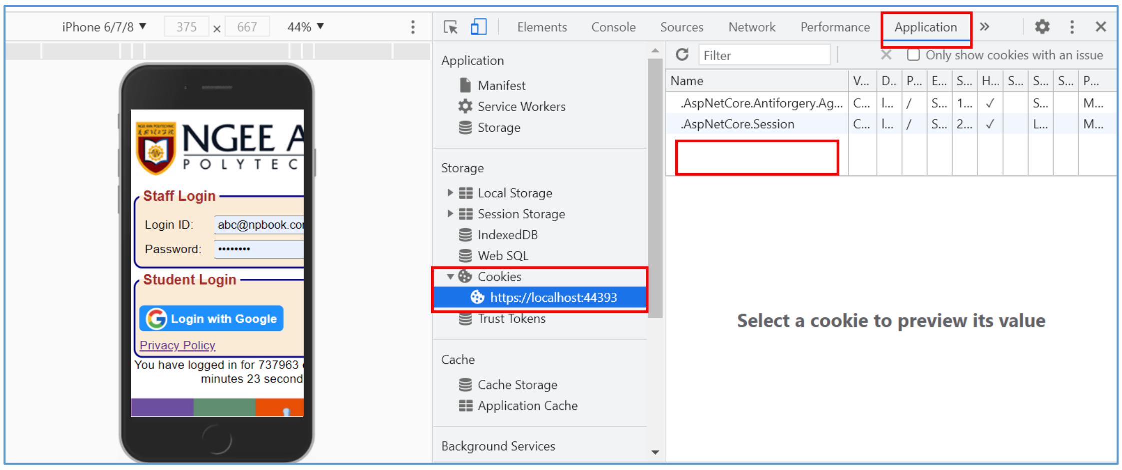
20. Modify and add the highlighted code in the ***LogOut()*** action method in **HomeController**. The code will clear all authentication cookies for the web app and hence logout the user's Google account.

```
public async Task<ActionResult> LogOut()
{
    // Clear authentication cookie
    await HttpContext.SignOutAsync(
          CookieAuthenticationDefaults.AuthenticationScheme);

    // Clear all key-value pairs stored in session state
    HttpContext.Session.Clear();

    // Go back to Home Page
    return RedirectToAction("Index");
}
```

[Note: The LogOut() action method is shared between Staff and Student Login. The HttpContext.SignOutAsyn() function will not cause any error even if there is no authentication cookie being saved for the web app.]

21. Test the page using Google Chrome, log in with your Google personal account and then log out. You should be re-directed to the home page and when you click at the "Sign in with Google" button again, it will prompt you to select a Google account.

22. Press function <F12> to open the Debugger of Google Chrome, Open the "Application" tab at select the Storage > Cookies field, you will see the 2 cookie chunks (.AspNetCore.CokkiesC1 and .AspNetCore.CokkiesC2) have been removed after you have logout successfully.



Name	V...	D..	P...	E...	S...	H...	S...	S...	S...	P...
.AspNetCore.Antiforgery.Ag...	C...	l...	/	S...	1...	✓		S...		M...
.AspNetCore.Session	C...	l...	/	S...	2...	✓		L...		M...

23. Go to the Book controller index by type URL: https://localhost:<PortNumber>/Book. You will be redirected to the Google Sign In page. This prove that the [Authorize] attribute added to the controller is working.

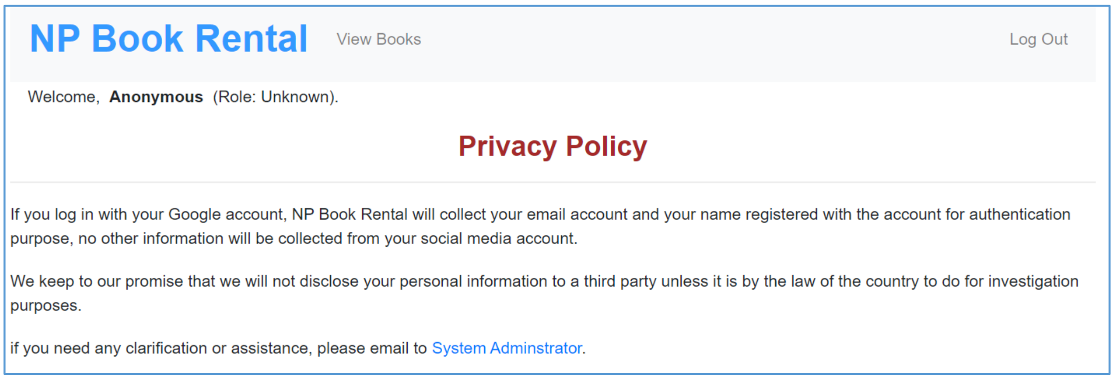
Part 4 – Privacy Protection Policy Page

When using single sign-on with social media account, it is important to inform users what information your app is obtaining from their account and assure that the information will not be disclosed to a third-party without their consent.

24. Open the view **Privacy.cshtml** from the folder Views\Home from Solution Explorer. Replace the code from Line 4 onward with the highlighted code as follow:

```
@{
    ViewData["Title"] = "Privacy Policy";
}
<h4 class="PageTitle">Privacy Policy</h4>
<hr />
<p>
If you log in with your Google account, NP Book Rental will collect your
email account and your name registered with the account for
authentication purpose, no other information will be collected from your
social media account.
</p>
<p>
We keep to our promise that we will not disclose your personal
information to a third party unless it is by the law of the country to do
for investigation purposes.
</p>
<p>
if you need any clarification or assistance, please email to <a
href="mailto:abc@npbook.com">System Adminstrator</a>.
</p>
```

25. Save the file and test the page by clicking the “Privacy Policy” link below “Sign in with Google” button:



Welcome, **Anonymous** (Role: Unknown).

Privacy Policy

If you log in with your Google account, NP Book Rental will collect your email account and your name registered with the account for authentication purpose, no other information will be collected from your social media account.

We keep to our promise that we will not disclose your personal information to a third party unless it is by the law of the country to do for investigation purposes.

if you need any clarification or assistance, please email to System Adminstrator.

CONCLUSION

- We have learnt how to configure a Google Single-Sign-On using OAuth 2.0.

Practical 13 – *Checklist*

No	Item	Yes/No
1	I understand how to configure a Single-Sign-On using OAuth 2.0 with Google account.	
2	I understand how the process of OAuth2 authorization and the use of access token/ID token in performing authentication and authorization.	