“Web API Authentication with Google and Authorization”

A Project Report
Submitted in partial fulfillment of the
Requirements for the award of the Degree of

MASTERS OF SCIENCE (INFORMATION TECHNOLOGY)

Submitted by

Kedar Sitaram Jadhav (Roll No. 30)

DEPARTMENT OF INFORMATION TECHNOLOGY
RAMNIRANJAN JHUNJHUNWALA COLLEGE
(Affiliated to University of Mumbai)
GHATKOPAR (WEST), 400086
MAHARASHTRA
2021-2022

DEPARTMENT OF INFORMATION TECHNOLOGY

CERTIFICATE

This is to certify that the project entitled, "Web API Authentication with Google and Authorization", is bonafied work of name bearing Seat. No: 30 submitted in partial fulfillment of the requirements for the award of degree of MASTERS OF SCIENCE in INFORMATION TECHNOLOGY from University of Mumbai.

Internal Guide

Coordinator

External Examiner

Date: 09-Apr-2022

College Seal

ACKNOWLEDGEMENT

I have great pleasure in presenting this project entitled "**Web API Authentication with Google and Authorization**" and I grab this opportunity to convey my immense regards to all people who with their invaluable contributions made this project successful.

It gives me great pleasure in presenting this Project Report. Its justification will never complete if I don't express my vote of thanks **RAMNIRANJAN JHUNJHUNWALA COLLEGE** AND **Dr. HIMANSHU DAWDA**.

I sincerely thank and express my profound gratitude to our Project Guide **Mrs. BHARTI BHOLE** for timely and prestigious guidance required for the project completion at each of the project development.

INDEX

1. Project Introduction

In this project I will show you how to create and build Web API and add authentication on it. Authentication is used to protect our applications and websites from unauthorized access and also, it restricts the user from accessing the information from tools like postman and fiddler. In this article, we will discuss basic authentication, how to call the API method using postman, and consume the API using jQuery Ajax.

Authentication schemes provide a secure way of identifying the calling user. Endpoints also checks the authentication token to verify that it has permission to call an API. Based on that authentication, the API server decides on authorizing a request.

The API authentication process validates the identity of the client attempting to make a connection by using an authentication protocol. The protocol sends the credentials from the remote client requesting the connection to the remote access server in either plain text or encrypted form. The server then knows whether it can give access to that remote client or not.
That's essentially what API authentication is all about. The system needs to make sure each end-user is properly validated. The system also wants to make sure the client system does not represent someone who has accidentally tried to access a service they are not entitled too, or worse—the system of a cybercriminal trying to hack into the system.

2. All About Web API

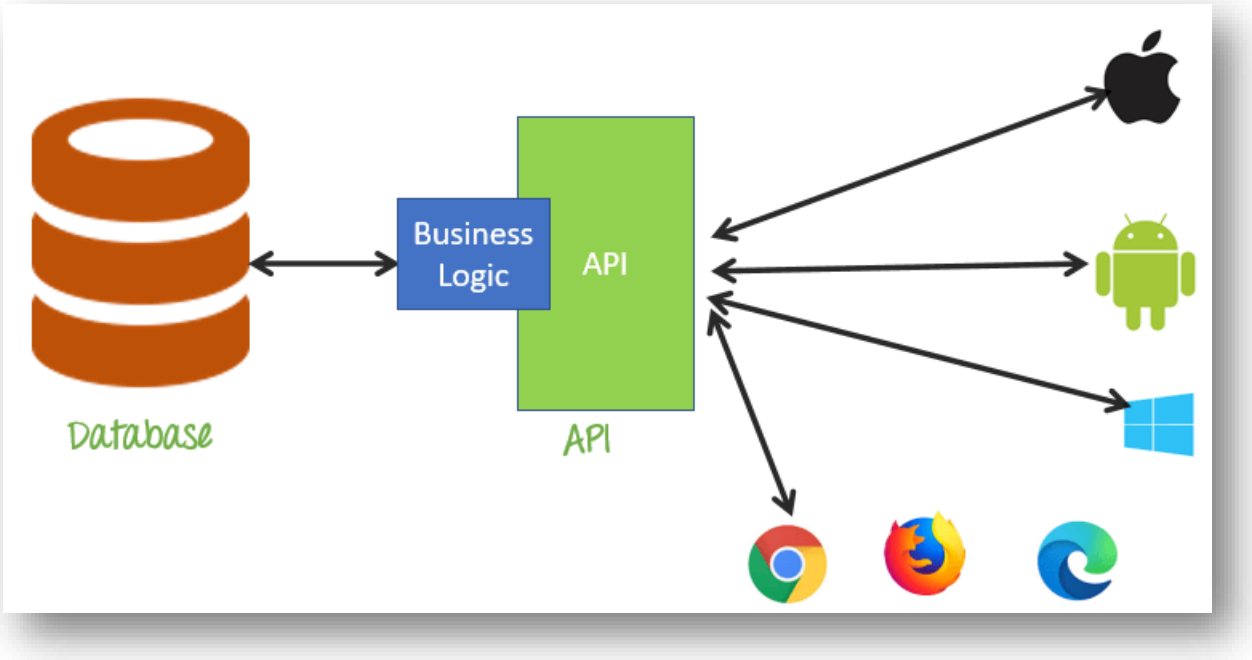
2.1.1 Definition of API

Application Programming Interface (API) is a software interface that allows two applications to interact with each other without any user intervention.

API is a collection of software functions and procedures. In simple terms, API means a software code that can be accessed or executed. API is defined as a code that helps two different software's to communicate and exchange data with each other.

It offers products or services to communicate with other products and services without having to know how they're implemented.

An API is a set of programming code that enables data transmission between one software product and another. It also contains the terms of this data exchange.



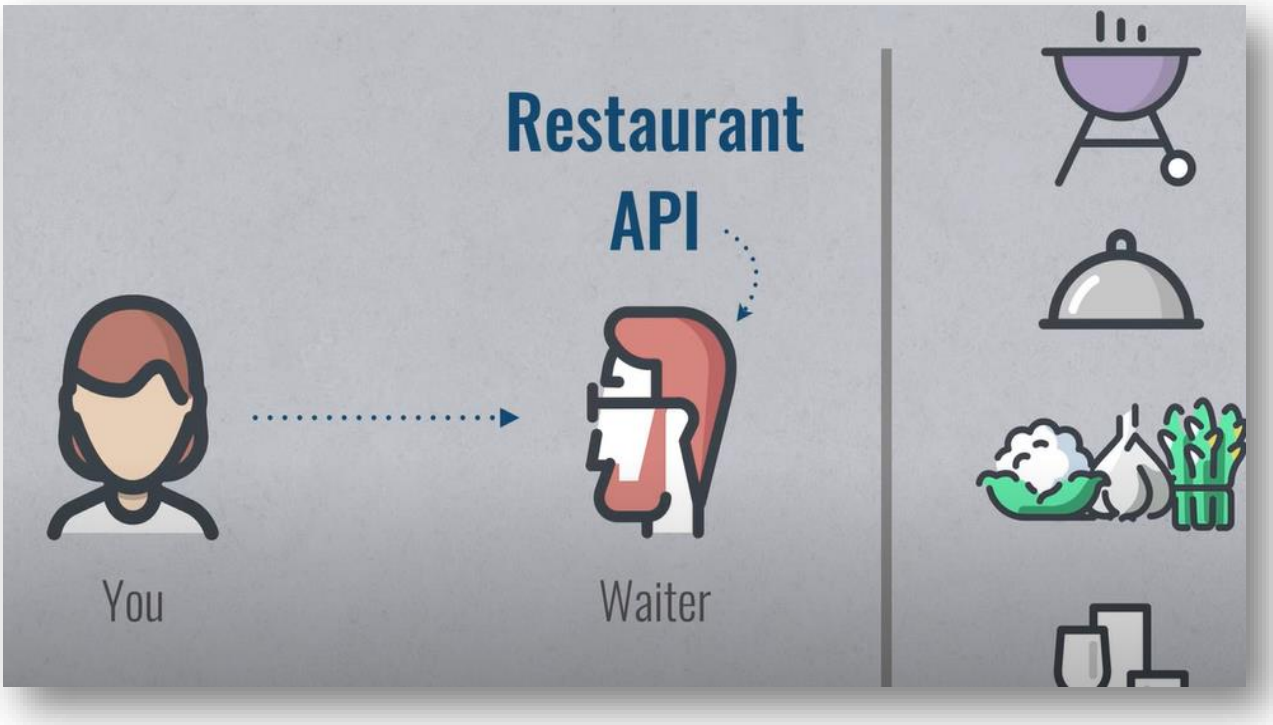
2.1.2 How Its Work

Let see how API works using simple daily life example. Imagine that you went to a restaurant to take lunch or dinner. The waiter comes to you gives you a menu card, and you will provide personalize it order like you want a veg sandwich but without onion.

After some time, you will get your order from the waiter. However, it is not that simple as it looks as there is some process that happens in between.

Here, the waiter plays an important part as you will neither go to the kitchen to collect your order nor will you tell the kitchen staff what you want all this done by the waiter.

API also does the same by taking your request, and just like the waiter tell the system what you want and give a response back to you.



2.2 Why would we need an API?

Here, are some reasons for using API:

- Application Programming Interface acronym API helps two different software's to communicate and exchange data with each other.
- It helps you to embed content from any site or application more efficiently.
- APIs can access app components. The delivery of services and information is more flexible.
- Content generated can be published automatically.
- It allows the user or a company to customize the content and services which they use the most.
- Software needs to change over time, and APIs help to anticipate changes.

Here are some Features of API:

- It offers a valuable service (data, function, audience,).
- It helps you to plan a business model.
- Simple, flexible, quickly adopted.
- Managed and measured.
- Offers great developer support.

Types of API

There are mainly Three main types of APIs:

1. Open APIs: These types of APIs are publicly available to use like OAuth APIs from Google. It has also not given any restriction to use them. So, they are also known as Public APIs.
2. Partner APIs: Specific rights or licenses to access this type of API because they are not available to the public.

3. Internal APIs: Internal or private. These APIs are developed by companies to use in their internal systems. It helps you to enhance the productivity of your teams.

Communication level of APIs:
Here, are some communication levels of APIS:

1. **High-Level APIs:**
   High-level APIs are those that we can generally use in REST form, where programmers have a high level of abstraction. These APIs mostly concerned about performing a limited functionality.

2. **Low-Level APIs:**
   This kind of APIs has a lower level of abstraction, which means they are more detailed. It allows the programmer to manipulate functions within an application module or hardware at a granular level.

2.3 Web API and types

A Web API is an application programming interface which is use either for web server or a web browser.
Two types of Web APIs are

1. **Server-side:**
   Server-side web API is a programmatic interface that consist of one or more publicly exposed endpoints to a defined request–response message system. It is typically expressed in JSON or XML

2. **Client-side:**
   A client-side web API is a programmatic interface helps to extend functionality within a web browser or other HTTP client.

Examples of web API:

1. Google Maps API's allow developers to embed Google Maps on webpages by using a JavaScript or Flash interface.
2. YouTube API allows developers to integrate YouTube videos and functionality into websites or applications.
3. Twitter offers two APIs. The REST API helps developers to access Twitter data, and the search API provides methods for developers to interact with Twitter Search.
4. Amazon's API gives developers access to Amazon's product selection.

2.4 API Specification and Protocol

APIs exchange commands and data, and this requires clear protocols and architectures - the rules, structures and constraints that govern an API's operation.
Today, there are three categories of API protocols or architectures: REST, RPC and SOAP.
These may be dubbed "formats," each with unique characteristics and tradeoffs and employed for different purposes.

1. **REST.** The representational state transfer (REST) architecture is perhaps the most popular approach to build APIs. REST relies on a client/server approach which separates front and back ends of the API, and provides considerable flexibility in development and implementation.
   REST is "stateless," which means the API stores no data or status between requests. REST supports caching, which stores responses for slow or non-time sensitive APIs.
   REST APIs, usually termed "RESTful APIs," also can communicate directly, or operate through intermediate systems such as API gateways and load balancers.

2. **RPC.** The remote procedural call (RPC) protocol is a simple means to send multiple parameters and receive results.
   RPC APIs invoke executable actions or processes, while REST APIs mainly exchange data or resources such as documents.
   RPC can employ two different languages, JSON and XML, for coding; these APIs are dubbed JSON-RPC and XML-RPC, respectively.

3. **SOAP.** The simple object access protocol (SOAP) is a messaging standard defined by the World Wide Web Consortium and broadly used to create web APIs, usually with XML.
   SOAP supports a wide range of communication protocols found across the internet such as HTTP, SMTP and TCP. SOAP is also extensible and style-independent, which allows developers to write SOAP APIs in varied ways and easily add features and functionality.

   Consider REST and SOAP. Both formats are designed to connect applications and mainly utilize HTTP protocols and commands such as GET, POST and DELETE.
   Both can use XML in requests and responses. However, SOAP depends on XML by design, while REST can also use JSON, HTML and plain text. SOAP is standardized with strict rules, while REST allows flexibility in its rules and is instead governed by architectures.
   SOAP is built from remote procedure calls, while REST is based on resources.
   Thus, both REST and SOAP exchange information, but do so in very different ways. SOAP is used when an enterprise requires tight security and clearly-defined rules to support more complex data exchanges and ability to call procedures.
   Developers frequently use SOAP for internal or partner APIs.
   REST is used for fast exchanges of relatively simple data.
   REST can also support greater scalability, supporting large and active user bases.

2.5 API Testing Tools

1. **Postman**
   Postman is a plugin in Google Chrome, and it can be used for testing API services.
   It is a powerful HTTP client to check web services. For manual or exploratory testing, Postman is a good choice for testing API.

2. **Ping API**
   Ping-API is API testing allows us to write test script in JavaScript and Coffee Script to test your APIs.
   It will enable inspecting the HTTP API call with a complete request and response data.

3. **vREST**
   vREST API tool provides an online solution for automated testing, mocking, automatic recording, and specification of REST/HTTP APIs/RESTful APIs.

3. Implementation of Web API

3.1 Web API Development tools

1. Microsoft Visual studio 2019
2. Microsoft SQL Server Management studio 2018

3.2 System Requirements

- CPU Requirements:
  - Minimum Dual Core 1.8 GHz CPU
- RAM Requirements:
  - Minimum 8 GB available RAM
- Hard Disk Requirements:
  - Minimum 4 GB available hard disk space
- Microsoft .NET Framework Requirements:
  - Microsoft .NET Framework 4.7.2 or greater
- Other Requirements:
  - Microsoft OLE DB Driver 18 for SQL Server
  - MVC application pools are running .NET Framework v4.0.
  - Securing each installation with SSL.
  - IIS must have the URL Rewrite Module installed.
  - IIS must have the HTTP Module installed.
  - Postman

3.3 Required References

All the reference are added in NuGet packet Manager

1. **Microsoft.EntityFrameworkCore**

   - Entity Framework Core is a modern object-database mapper for .NET. It supports LINQ queries, change tracking, updates, and schema migrations. EF Core works with SQL Server, Azure SQL Database, SQLite, Azure Cosmos DB, MySQL, PostgreSQL, and other databases through a provider plugin API.
   - Commonly Used Types:
     1. Microsoft.EntityFrameworkCore.DbContext
     2. Microsoft.EntityFrameworkCore.DbSet

2. **Microsoft.EntityFrameworkCore.SqlServer**

   - Microsoft SQL Server database provider for Entity Framework Core.

3. **Microsoft.EntityFrameworkCore.Tools**

   - Entity Framework Core Tools for the NuGet Package Manager Console in Visual Studio.
   - Enables these commonly used commands:
     Add-Migration

4. **Microsoft.EntityFrameworkCore.Tools**

   - ASP.NET Core middleware that enables an application to receive an OpenID Connect bearer token.

5. **Microsoft.AspNetCore.Identity**

   - ASP.NET Core Identity is the membership system for building ASP.NET Core web applications, including membership, login, and user data. ASP.NET Core Identity allows you to add login features to your application and makes it easy to customize data about the logged in user.

6. **Microsoft.Owin.Security**

   - Common types which are shared by the various authentication middleware components.

7. **Microsoft.Owin.Security.Google**

   - Contains middlewares to support Google's OAuth 2.0 authentication workflow.

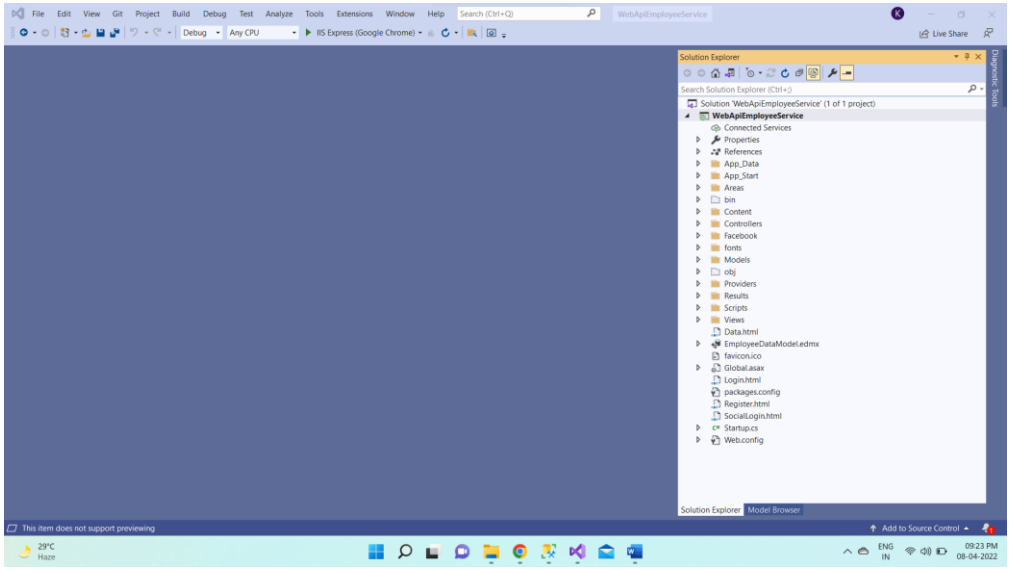
8. **Microsoft.Owin.Security.MicrosoftAccount**

   - Middleware that enables an application to support the Microsoft Account authentication workflow.

3.4 Create API Module

We will create our Web API using Visual Studio 2019. We can download the free community version from the Microsoft official site.

1. Open Visual Studio 2019 and create a new project.
2. Select the “Asp.Net Web application (.Net Framework)” template and click on Next.
3. Provide Project name and location.
4. Select Target Framework and click on Create button.
5. Member API is created. See below project structure.



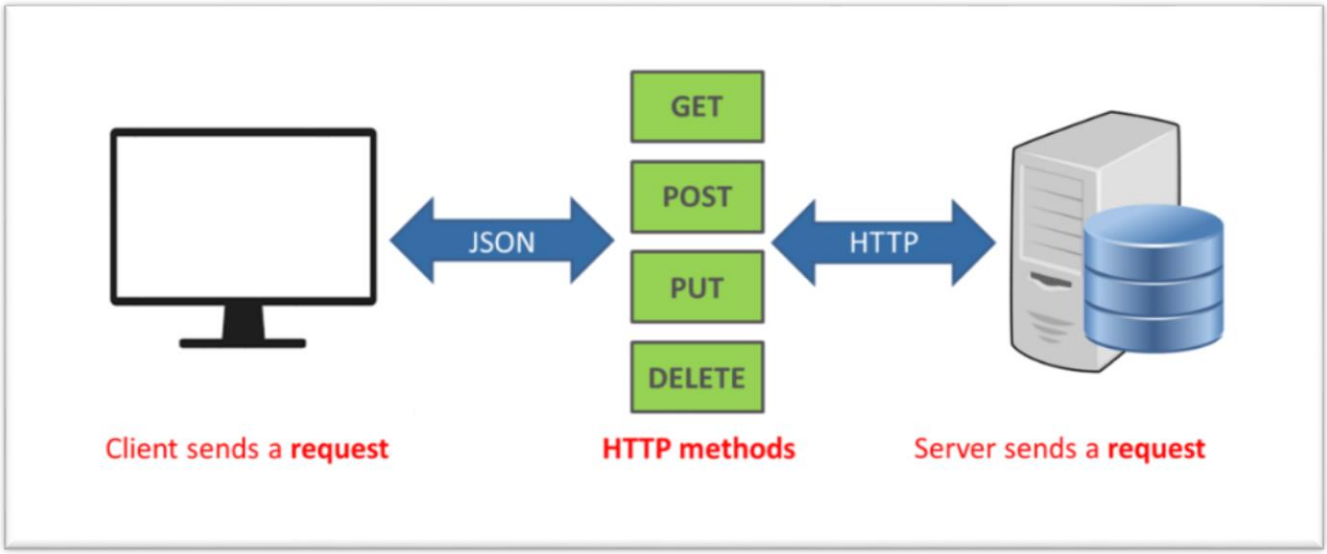
By Default, Weather API executed and displays output using Swagger. I hope you are aware of Swagger.

Simply put, Swagger is open-source software tool to design, build, document, and use RESTful Web API.

Web API is mostly used for CRED (Create, Read, EDIT, DELETE) operations. It follows HTTP verbs for these operations.

Method	Meaning
GET	Read data
POST	Insert data
PUT or PATCH	Update data, or insert if a new id
DELETE	Delete data

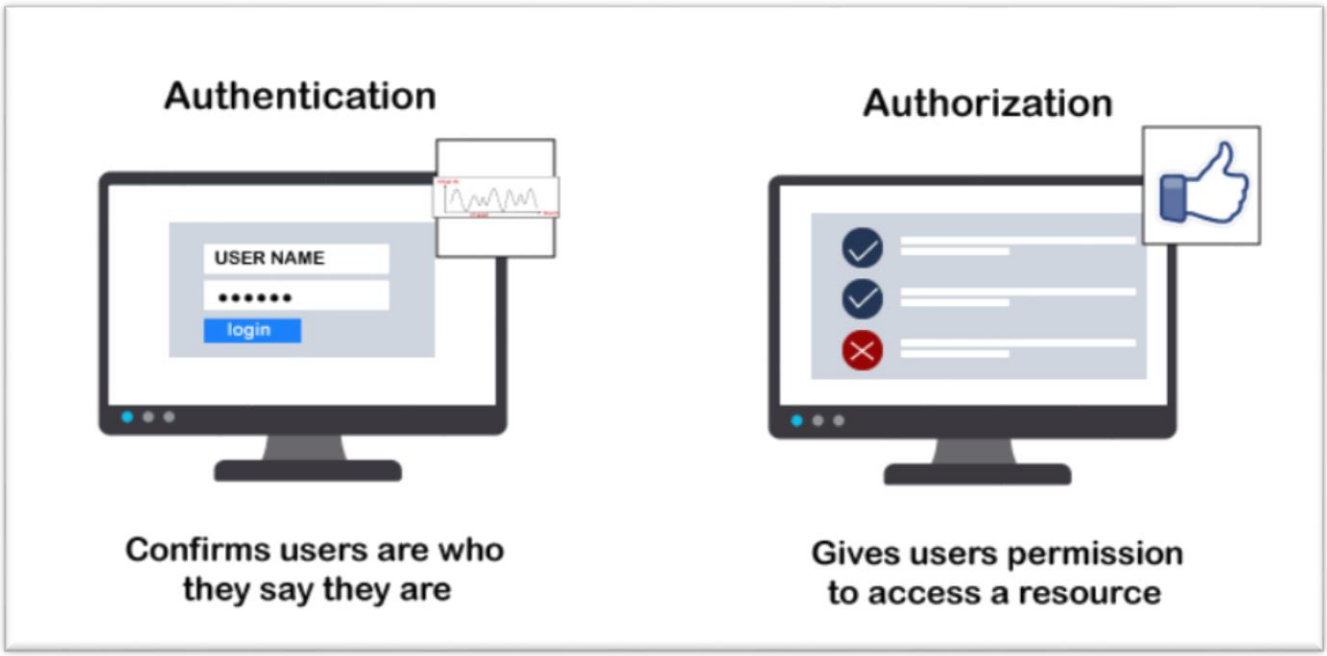
- HTTP GET – Read Operation
- HTTP POST – Add/Create Operation
- HTTP PUT – Update Operation
- HTTP DELETE – Delete Operation



4. Authentication in Web API

4.1 Web API Authentication and types

Authentication is when an entity proves an identity. In other words, Authentication proves that you are who you say you are. This is like having a driver license which is given by a trusted authority that the requester, such as a police officer, can use as evidence that suggests you are in fact who you say you are.
Authorization is an entirely different concept and in simple terms, Authorization is when an entity proves a right to access. In other words, Authorization proves you have the right to make a request. Consider the following - You have a working key card that allows you to open only some doors in the work area, but not all of them.



There are Most Used Authentication Methods
HTTP Authentication Schemes (Basic & Bearer)
The HTTP Protocol also defines HTTP security auth schemes like:

1. **Basic**
   - HTTP Basic Authentication is rarely recommended due to its inherent security vulnerabilities.

- This is the most straightforward method and the easiest. With this method, the sender places a username: password into the request header. The username and password are encoded with Base64, which is an encoding technique that converts the username and password into a set of 64 characters to ensure safe transmission.

2. Bearer

- Bearer authentication (also called token authentication) is an HTTP authentication scheme that involves security tokens called bearer tokens.
- The name "Bearer authentication" can be understood as "give access to the bearer of this token." The bearer token allowing access to a certain resource or URL and most likely is a cryptic string, usually generated by the server in response to a login request.

3. OAuth

- The previous versions of this spec, OAuth 1.0 and 1.0a, were much more complicated than OAuth 2.0. The biggest change in the latest version is that it's no longer required to sign each call with a keyed hash. The most common implementations of OAuth use one or both of these tokens instead:
- **access token**: sent like an API key, it allows the application to access a user's data; optionally, access tokens can expire.
- **refresh token**: optionally part of an OAuth flow, refresh tokens retrieve a new access token if they have expired. OAuth2 combines Authentication and Authorization to allow more sophisticated scope and validity control.

4. External Authentication

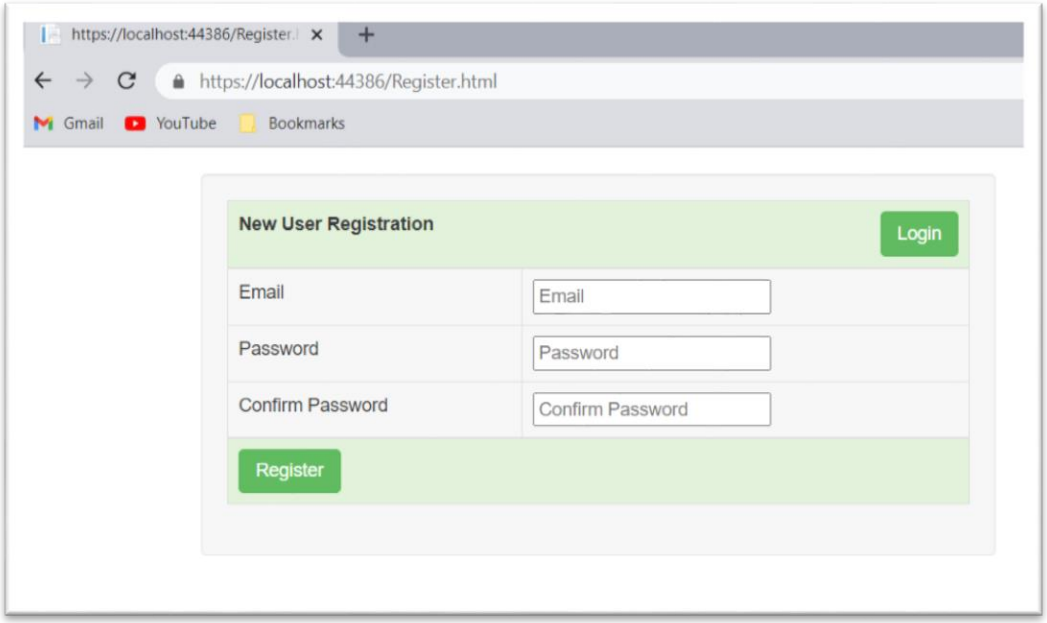
- Google APIs use the OAuth 2.0 protocol for authentication and authorization. Google supports common OAuth 2.0 scenarios such as those for web server, client-side, installed, and limited-input device applications.
- Visual Studio and ASP.NET 4.7.2 make integration with external authentication services easier for developers by providing built-in integration for the following authentication services:
  1. Facebook
  2. Google

3. Microsoft Accounts (Windows Live ID accounts)
4. Twitter

4.2 New User Registration

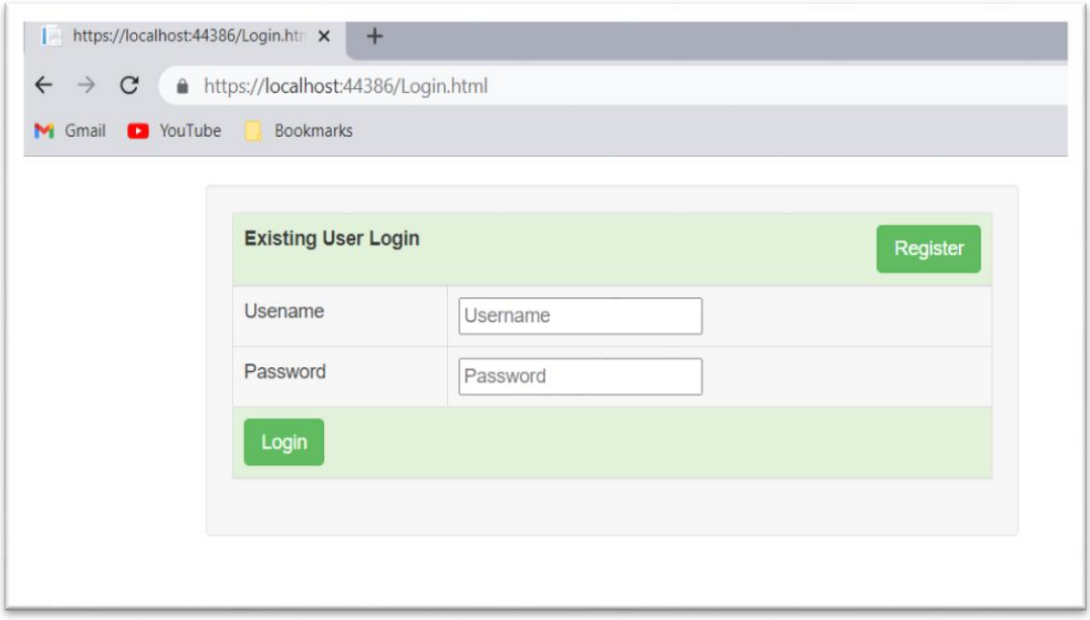
To register a new user with the API, follow these steps:

- Run the application, will open login page for creating new user. Provide user email id and password. Then click on Register button.

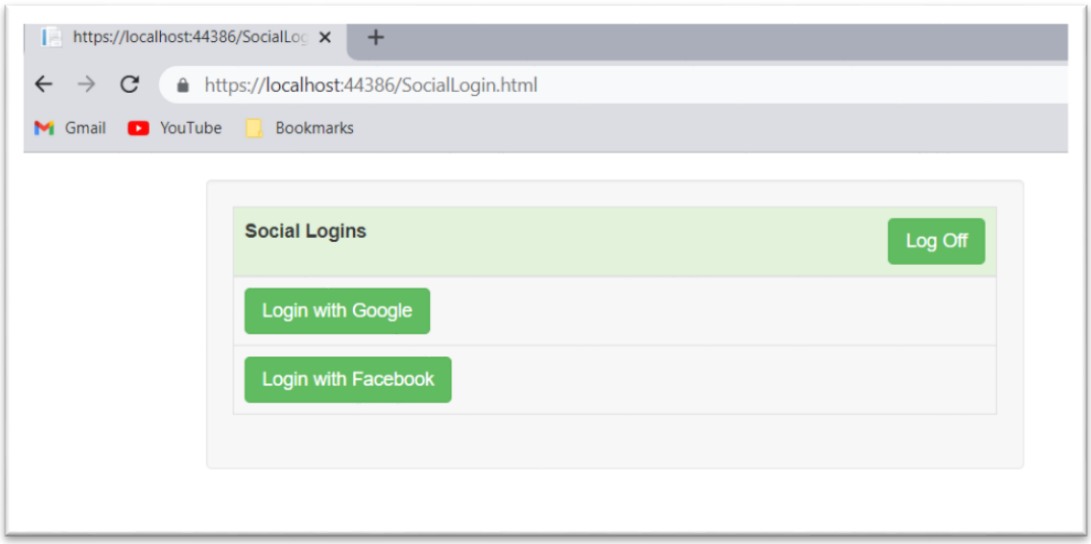


4.2 New or Existing User Login

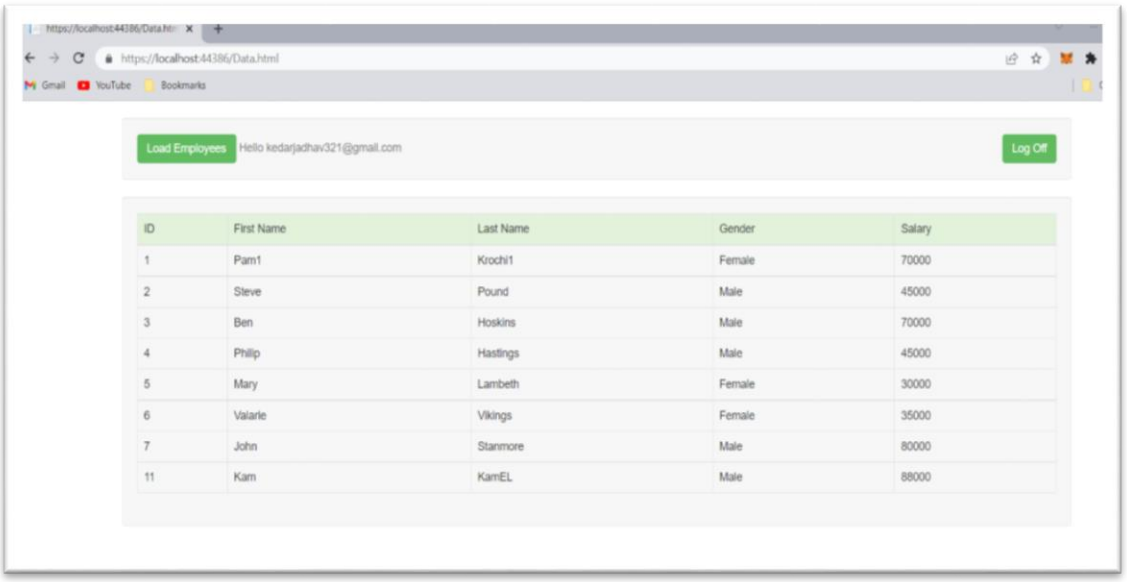
- Login with newly created user. Provide user email id and password. Then click on Register button.



- After login it will redirect to social login page to login with google credential or with Facebook credential.



- After successful login will redirect to Employee data page. It displays all the records.



4.3 Validate API Using JSON Web Token

JSON Web Token

- JSON Web Token (JWT) is an open standard (RFC 7519) that defines a compact and self-contained way for securely transmitting information between parties as a JSON object.
- This information can be verified and trusted because it is digitally signed. JWTs can be signed using a secret (with the **HMAC** algorithm) or a public/private key pair using **RSA** or **ECDSA**.

When should you use JSON Web Tokens?

Here are some scenarios where JSON Web Tokens are useful:

1. **Authorization**: This is the most common scenario for using JWT. Once the user is logged in, each subsequent request will include the JWT, allowing the user to access routes, services, and resources that are permitted with that token. Single Sign On is a feature that widely uses JWT nowadays, because of its small overhead and its ability to be easily used across different domains.

2. **Information Exchange**: JSON Web Tokens are a good way of securely transmitting information between parties. Because JWTs can be signed—for example, using public/private key pairs—you can be sure the senders are who they say they are. Additionally, as the signature is calculated

using the header and the payload, you can also verify that the content hasn't been tampered with.

JSON Web Token structure

In its compact form, JSON Web Tokens consist of three parts separated by dots (.), which are:

- **Header**
  The header typically consists of two parts: the type of the token, which is JWT, and the signing algorithm being used, such as HMAC SHA256 or RSA.

```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

- **Payload**
  The second part of the token is the payload, which contains the claims. Claims are statements about an entity (typically, the user) and additional data.

```
{
  "sub": "1234567890",
  "name": "John Doe",
  "admin": true
}
```

- **Signature**
  To create the signature part, you have to take the encoded header, the encoded payload, a secret, the algorithm specified in the header, and sign that.

  The signature is used to verify the message wasn't changed along the way, and, in the case of tokens signed with a private key, it can also verify that the sender of the JWT is who it says it is.

```
HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  secret)
```

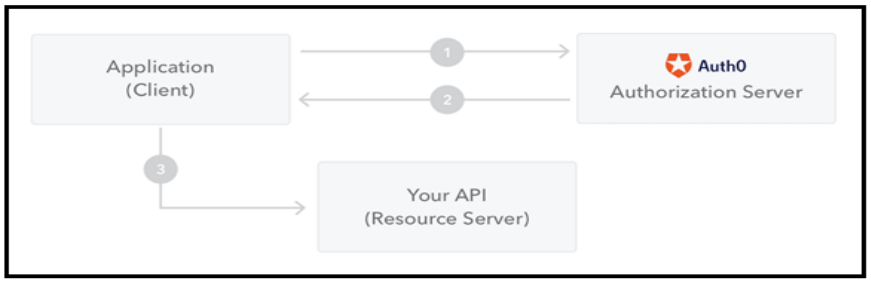
  Therefore, a JWT typically looks like the following.
  xxxxx.yyyyy.zzzzz

4.4 How do JSON Web Tokens work

- In authentication, when the user successfully logs in using their credentials, a JSON Web Token will be returned.
- Since tokens are credentials, great care must be taken to prevent security issues. In general, you should not keep tokens longer than required.
- Whenever the user wants to access a protected route or resource, the user agent should send the JWT, typically in the **Authorization** header using the **Bearer** schema. The content of the header should look like the following:

```
Authorization: Bearer <token>
```

- This can be, in certain cases, a stateless authorization mechanism. The server's protected routes will check for a valid JWT in the Authorization header, and if it's present, the user will be allowed to access protected resources. If the JWT contains the necessary data, the need to query the database for certain operations may be reduced, though this may not always be the case.
- If the token is sent in the Authorization header, Cross-Origin Resource Sharing (CORS) won't be an issue as it doesn't use cookies.
- The following diagram shows how a JWT is obtained and used to access APIs or resources:

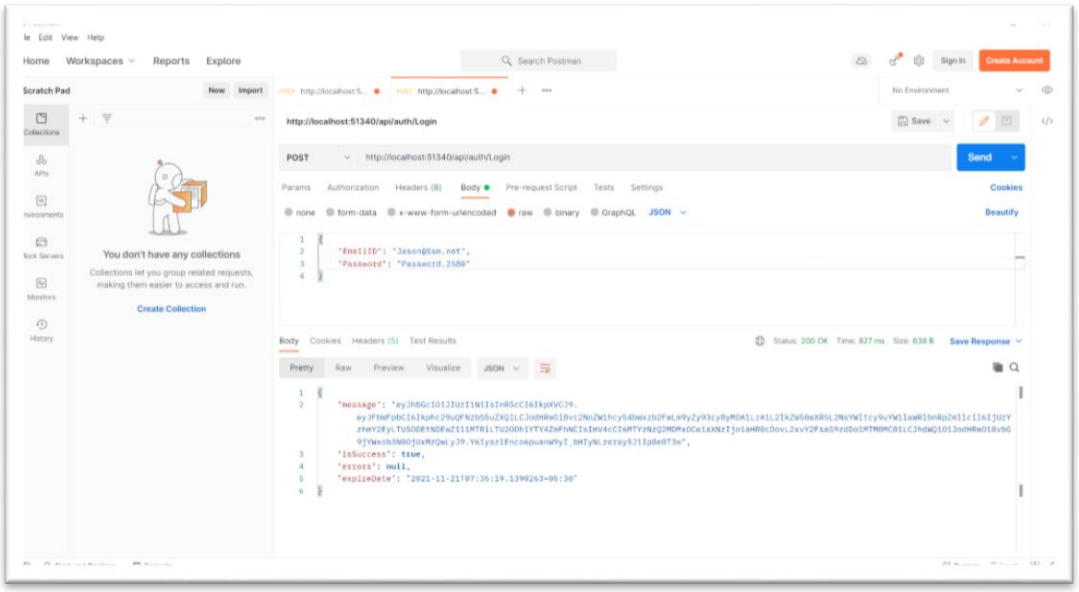


To Login with a user and generate Token with the API, follow these steps:

1. Open a new request tab by clicking the plus (+) button at the end of the tabs.
2. Change the http request method to "POST" with the dropdown selector on the left of the URL input field.

3. In the URL field enter the address to the register route of your local API - http://localhost:12345/api/auth/Login.
4. Select the "Body" tab below the URL field, change the body type radio button to "raw", and change the format dropdown selector to "JSON (application/json)".
5. Enter a JSON object containing the required user properties in the "Body" textarea,
   e.g:
   {
"EmailID": "",
"Password": " ",
   }

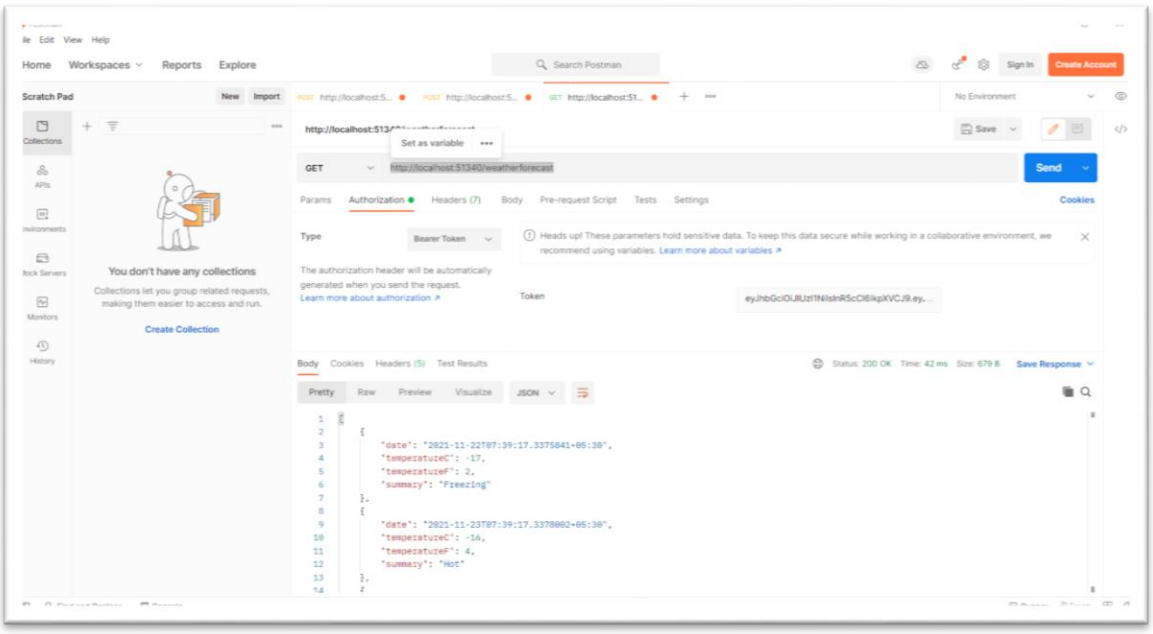
6. Click the "Send" button, you should receive a "200 OK" response with an JSON Token in the response body.



To Get the data using generated Token with the API, follow these steps:

1. Open a new request tab by clicking the plus (+) button at the end of the tabs.
2. Change the http request method to "GET" with the dropdown selector on the left of the URL input field.
3. In the URL field enter the address to the register route of your local API - http://localhost:12345/weatherforecast.
4. Select the "Authorization" tab below the URL field, Select Type to "Bearer Token", and in Token filed add the generated new token

5. Click the "Send" button, you should receive a "200 OK" response with API data.

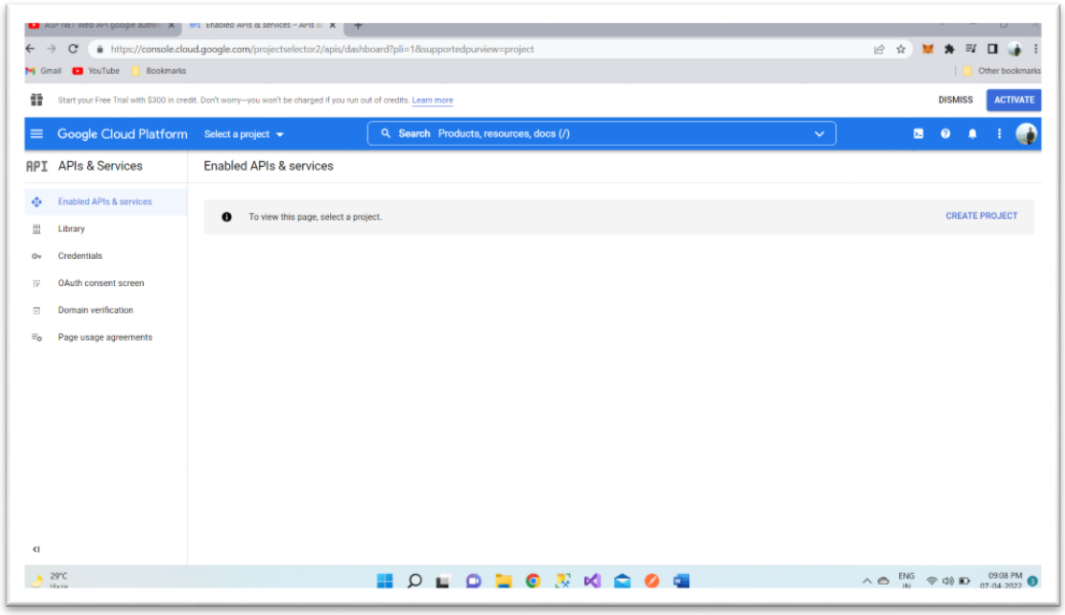


4.5 Web API Google Authentication

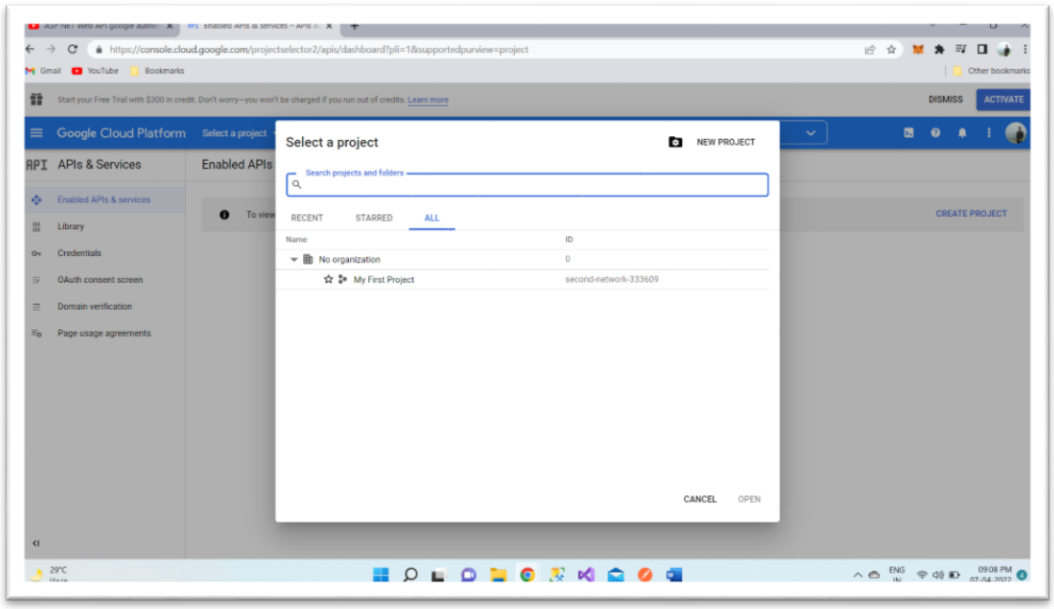
To use Google account for authentication, we will have to first register our application with Google. Here are the steps to register your application with Google. Once we successfully register our application with Google, we will be given a **Client ID** and **Client Secret**. We need both of these for using Google authentication with our Web API service.

1. : To register your application, go to https://console.developers.google.com

2. Login with your GMAIL account. Click on Credentials link on the left, and then create a new project, by clicking on "Create Project" button.

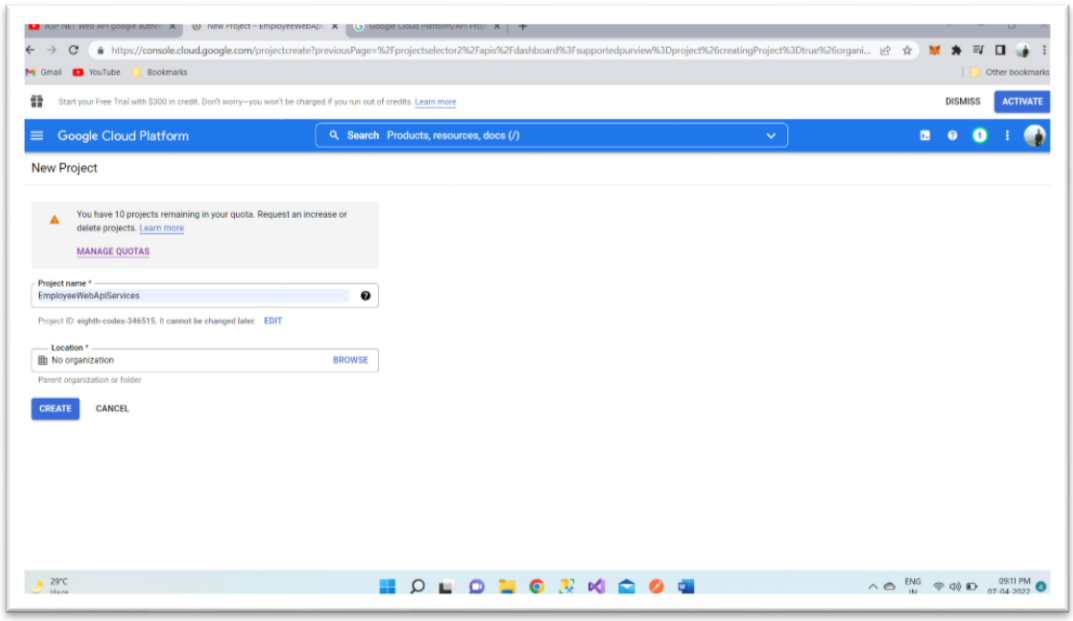


3. Name your project "Test Project" and click "CREATE" button.

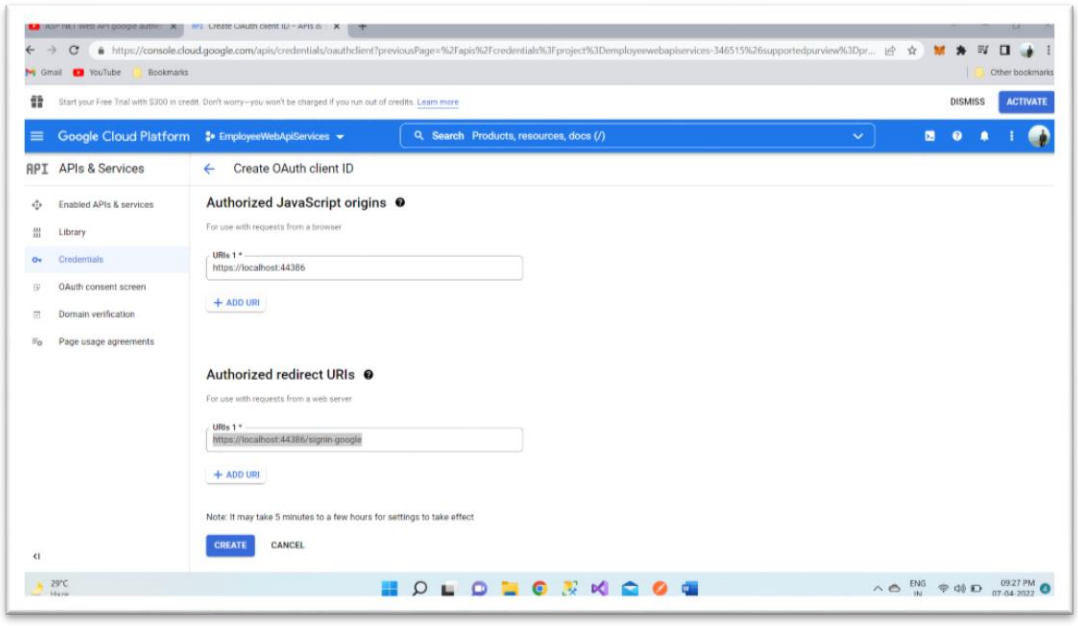


4. The new project will be created. Click on "OAuth consent screen". In the "Product name shown to users" textbox type "Test Project" and

click "CREATE" button.



5. The changes will be saved and you will be redirected to "Credentials" tab. If you are not redirected automatically, click on the "Credentials" tab and you will see "Create Credentials" dropdown button. Click on the button, and select "OAuth client ID" option

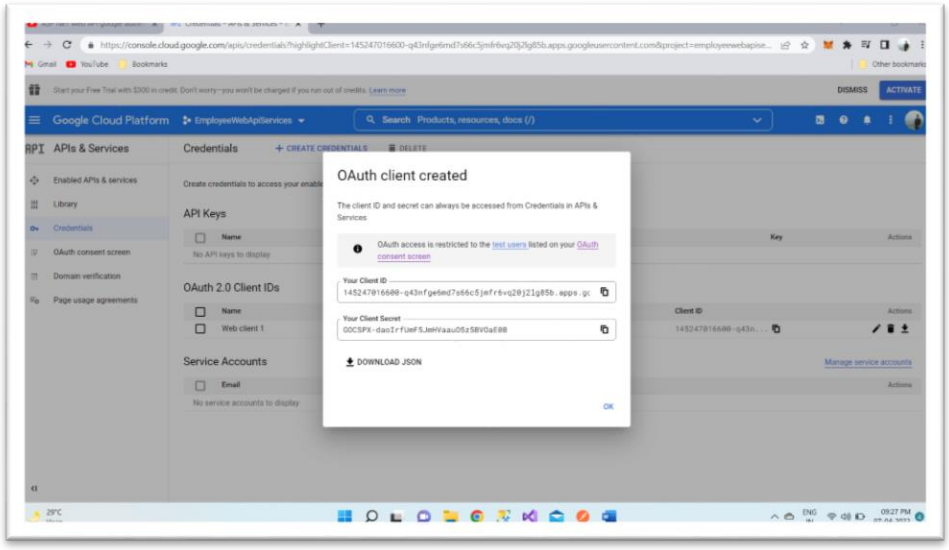


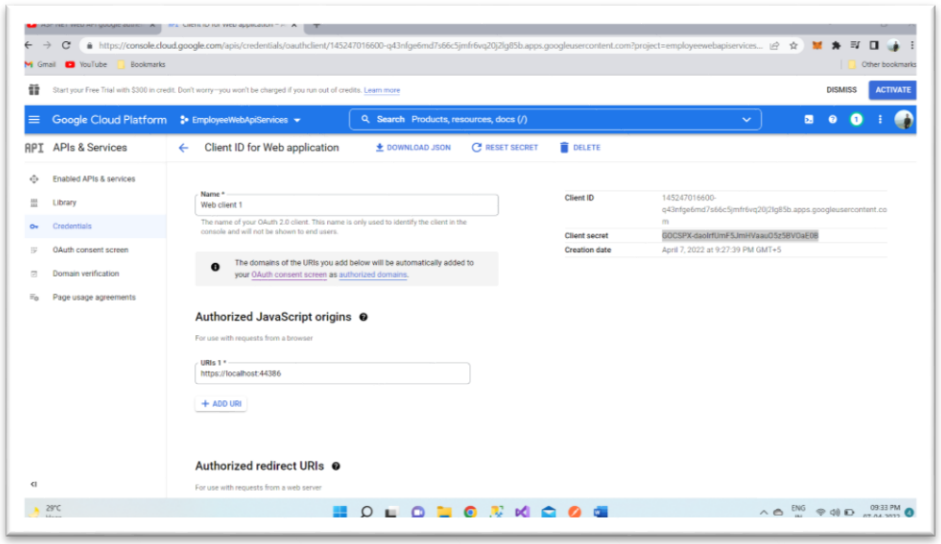
6. On the next screen,

    - Select "Web application" radio button.

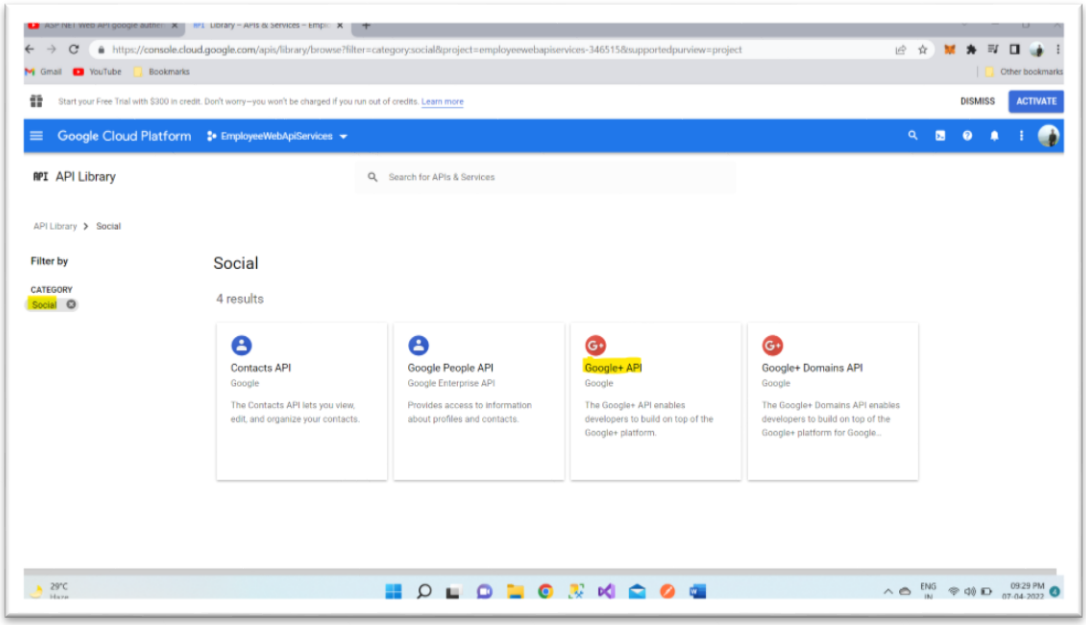
- Type "Web client 1" in the "Name" textbox.
- In the "Authorized JavaScript origins" textbox type in the URI of your application. I have my web api application running at http://localhost:44386
- In the "Authorized redirect URIs" textbox type in the redirect URI i.e the path in our application that users are redirected to after they have authenticated with Google. I have set it to http://localhost:44386/signin-google
- Click the "Create" button

You will see a popup with OAuth client ID and client secret. Make a note of both of them. We will need both of these later

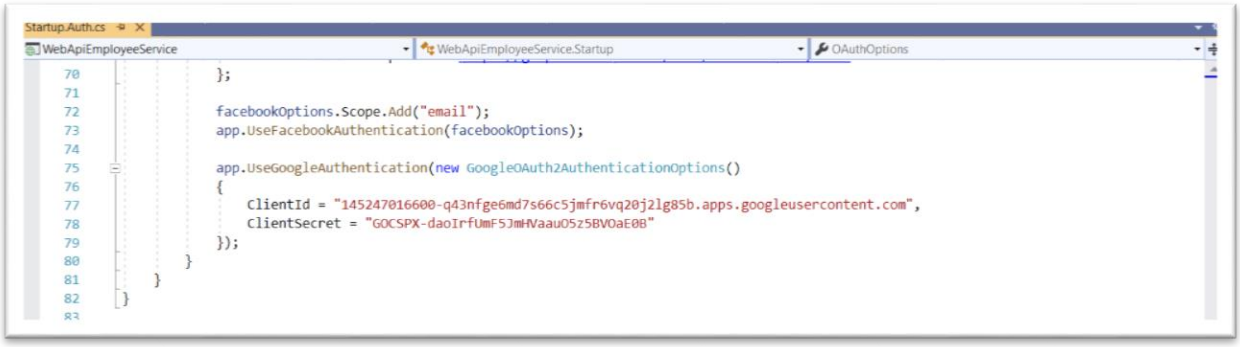




7. Enable Google+ API service. To do this click on "Library" link on the left-hand pane. Under "Social APIs" click on "Google+ API" link and click "Enable" button.



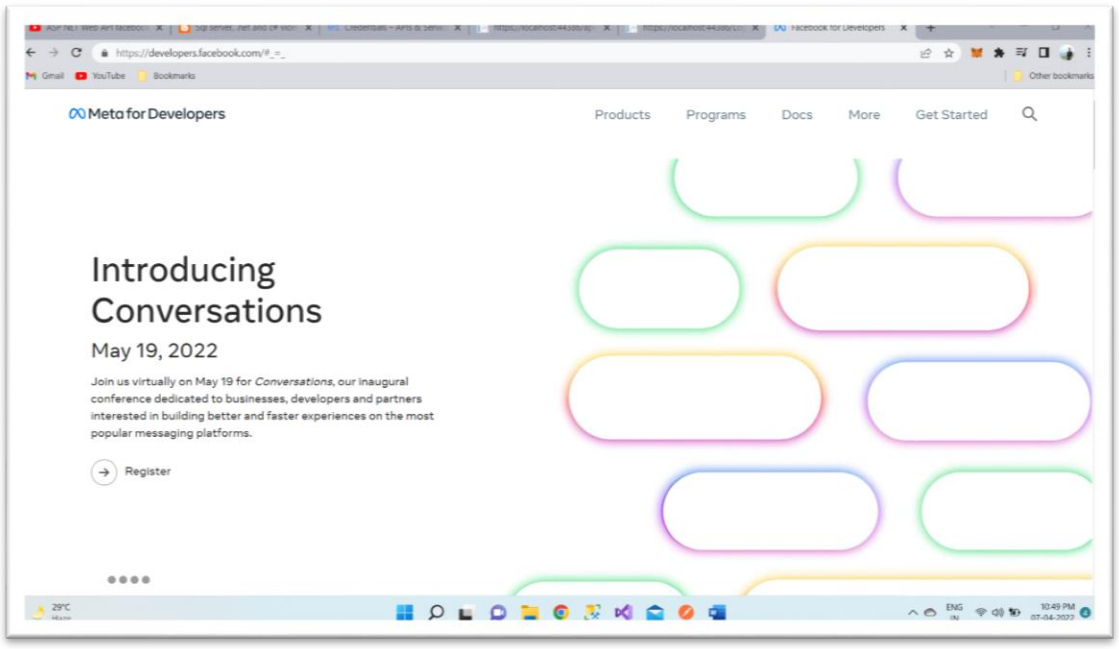
8. In Startup.Auth.cs file in App_Start folder un-comment the following code block, and include ClientId and ClientSecret that we got after registering our application with Google.



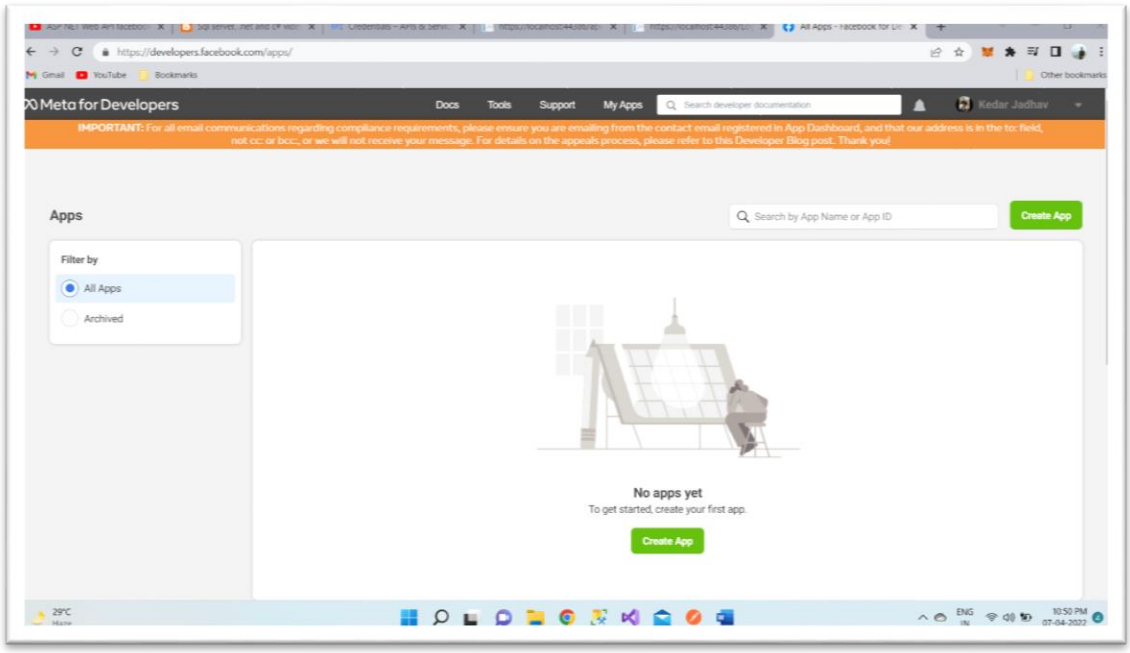
4.6 Web API Facebook Authentication

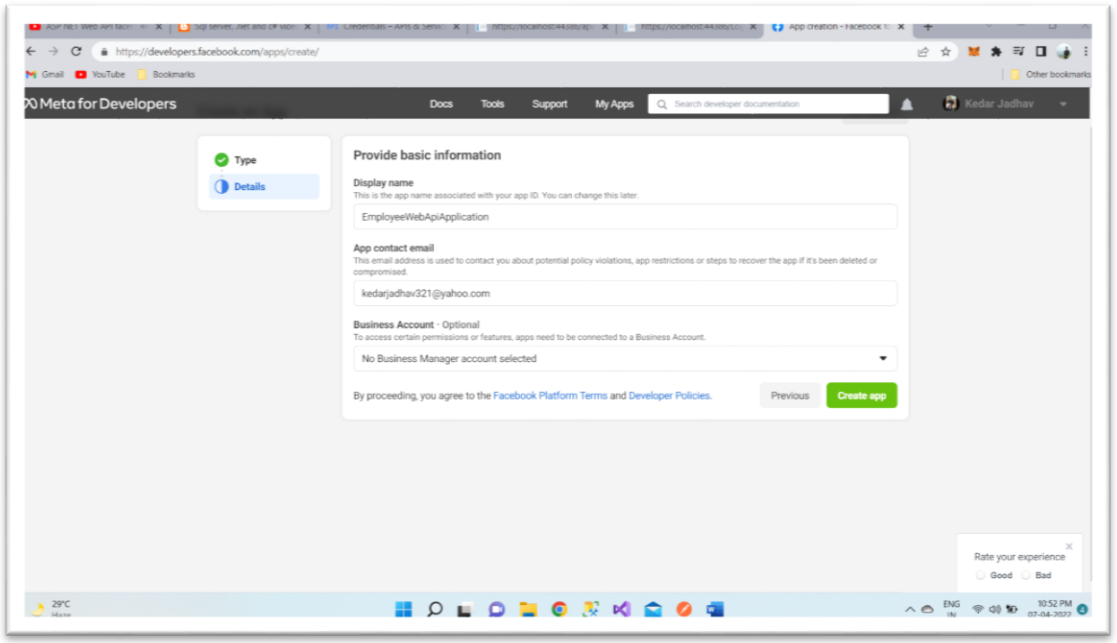
To use Facebook account for authentication, we will have to first register our application with Facebook. Here are the steps to register your application with Facebook. Once we successfully register our application, we will be given a **Client ID** and **Client Secret**. We need both of these for using Facebook authentication with our Web API service.

1. To register your application, go to https://developers.facebook.com/



2. Login with your Facebook account. Click "Create App" button

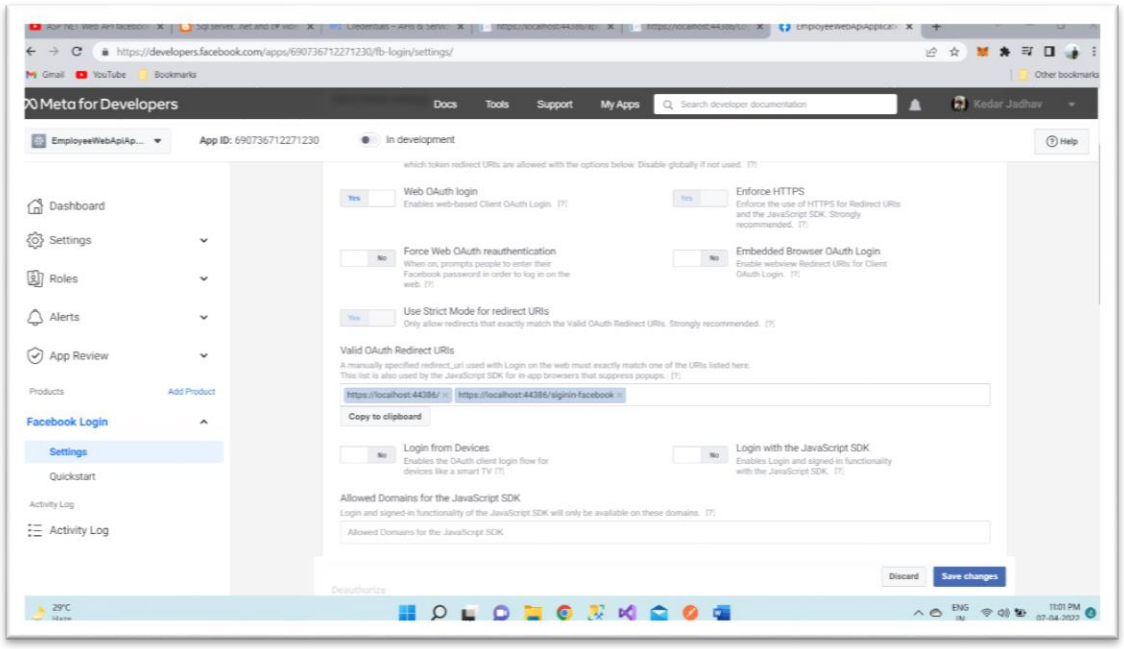




- Provide a name for the App in the "Display Name" textbox. I named it TestApp
- Provide your email in the "Contact Email" textbox
- Select a Category. I selected "Education"
- Finally, click "Create App ID" button

3. Next, we need to make the App that we just created public. To make the App, public, click on "App Review" menu item on the left and then click

the Yes|No button.

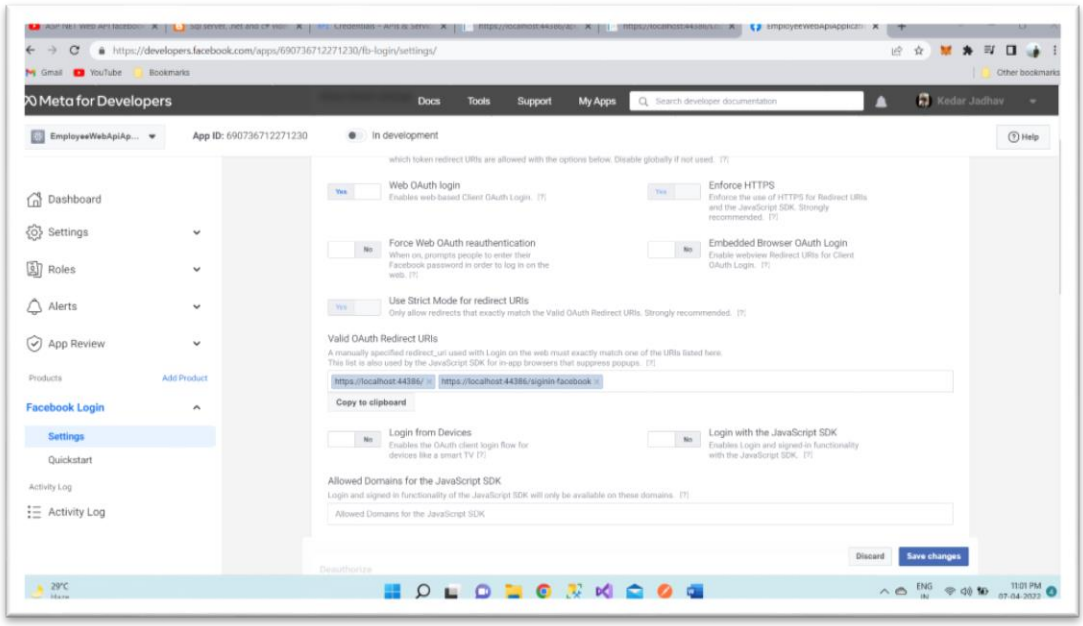


4. Next, we need to add "Facebook Login" product to our App. To do this click "Add Product" menu item on the left and click "Get Started" button against "Facebook Login"

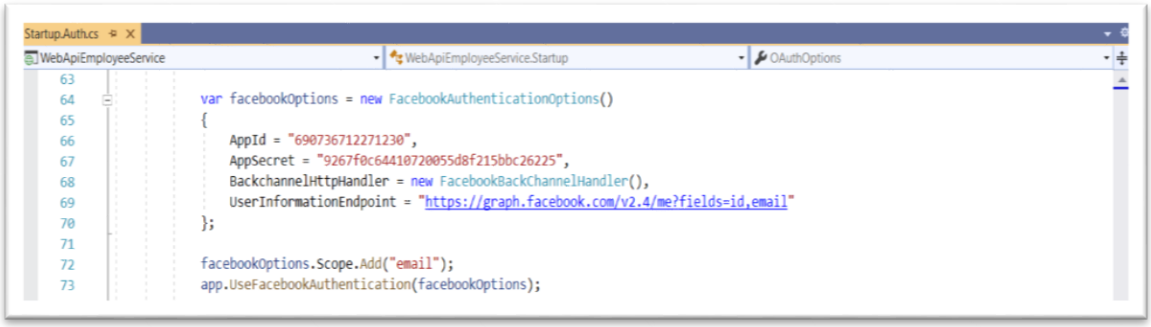
5. In "Valid OAuth redirect URIs" textbox, include the URI of your application. I have my web API application running at http://localhost:44386

   In the same textbox we also need to include the redirect URI i.e., the path in our application that users are redirected to after they have authenticated

with Facebook. I have set it to http://localhost:44386/signin-facebook



6. In Startup.Auth.cs file in App_Start folder include the following code block. Use the AppID and AppSecret that we got after registering our application with Facebook.

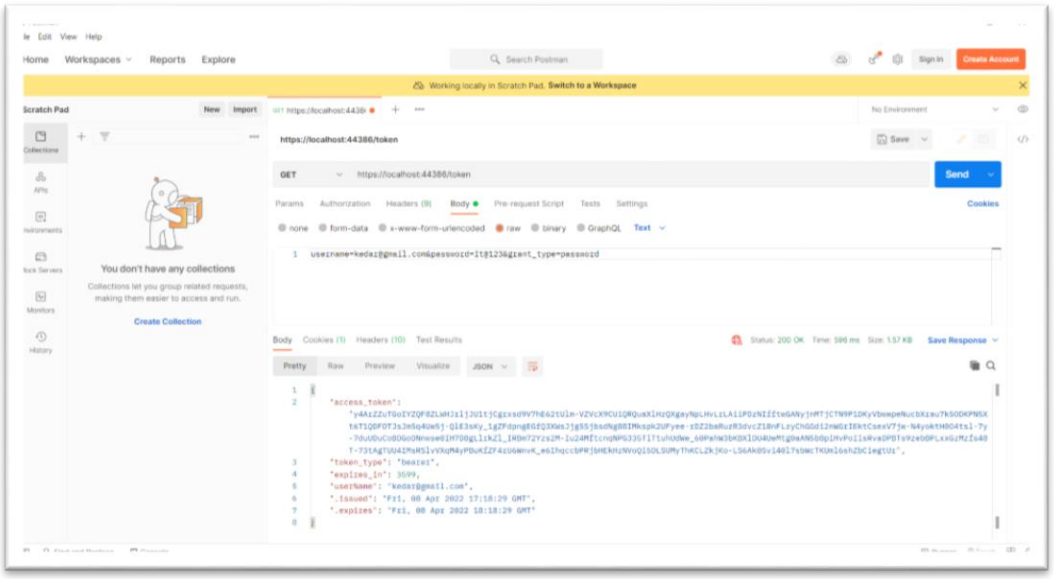


4.6 Using Postman – to view, add, update and delete data

1. To Get the data access generate token, follow these steps:

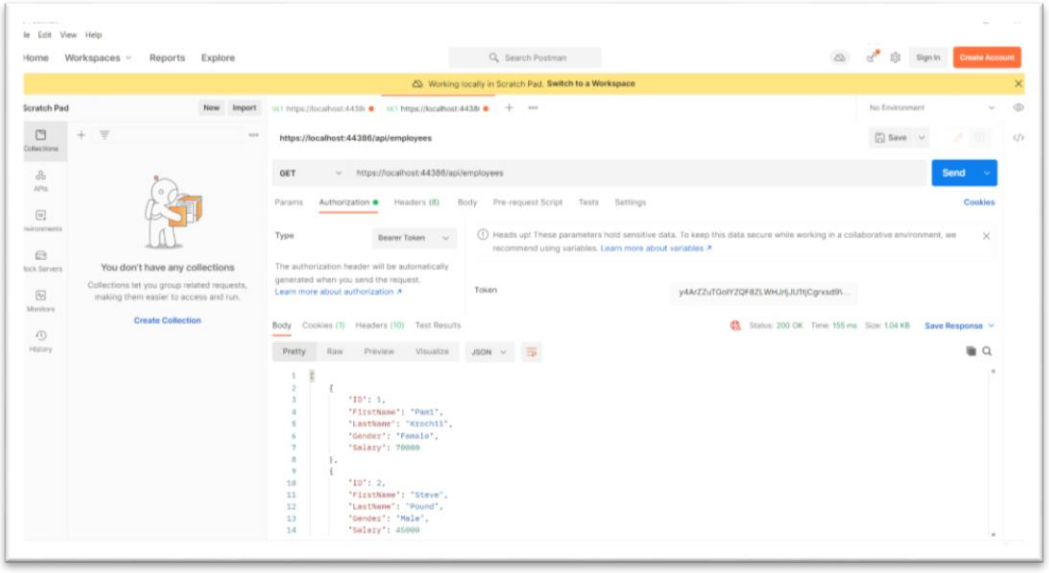
    1. Open a new request tab by clicking the plus (+) button at the end of the tabs.
    2. Change the http request method to "GET" with the dropdown selector on the left of the URL input field.
    3. In the URL field enter the address to the register route of your local API - https://localhost:44386/token

4. Select the "Body" tab below the URL field, Select Type to "Raw", and in Token filed add the generated new token
5. Click the "Send" button, you should receive a "200 OK" response with API data.

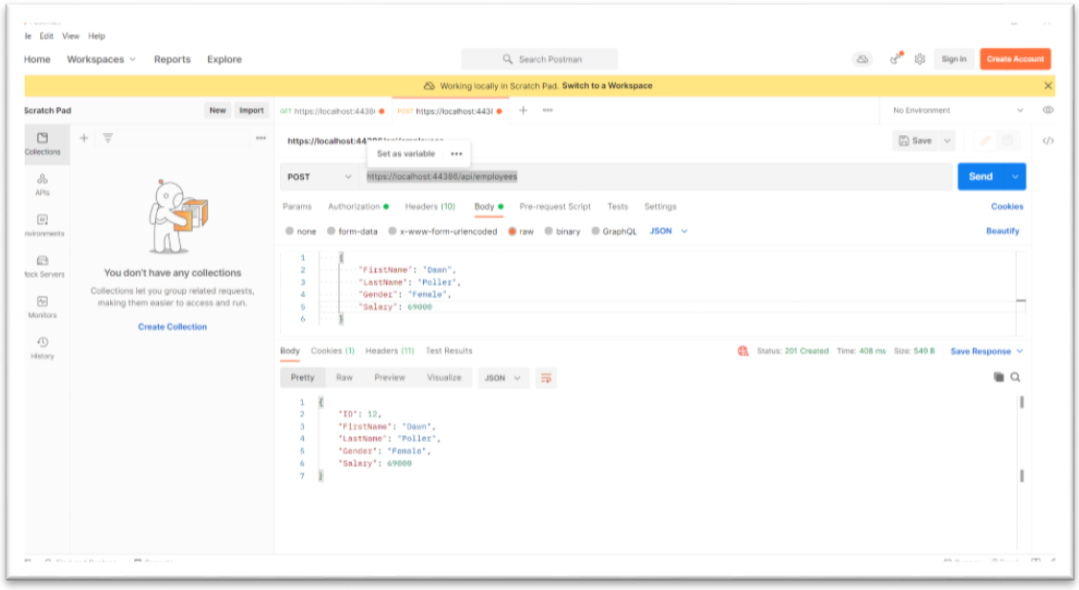


2. To View data using generated Token with the API, follow these steps:

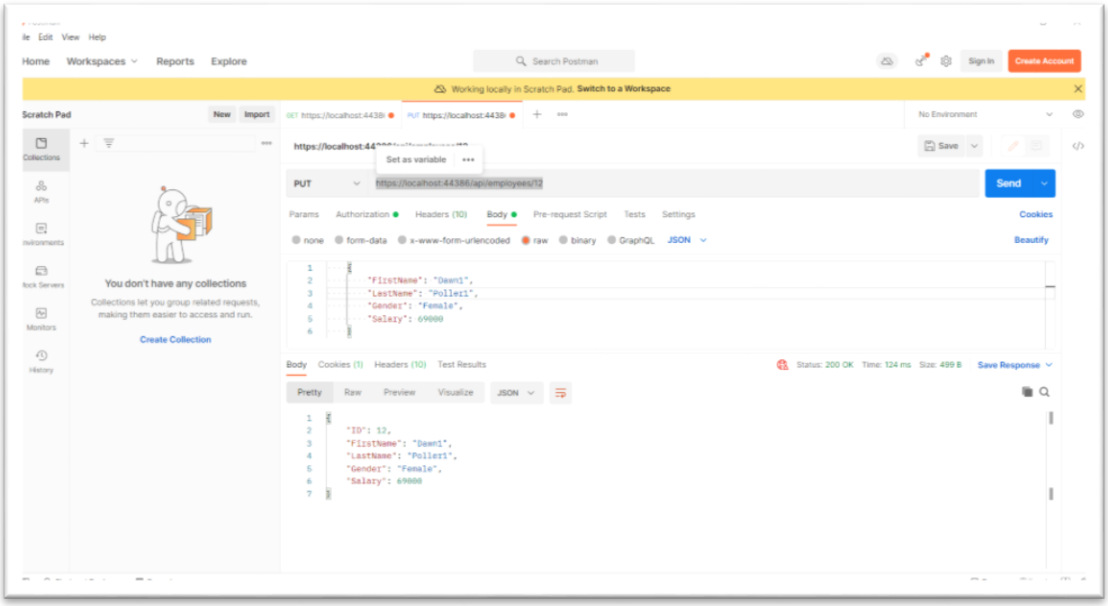
    1. Open a new request tab by clicking the plus (+) button at the end of the tabs.
    2. Change the http request method to "GET" with the dropdown selector on the left of the URL input field.
    3. In the URL field enter the address to the register route of your local API - https://localhost:44386/api/employees
    4. Select the "Authorization" tab below the URL field, Select Type to "Bearer Token", and in Token filed add the generated new token
    5. Click the "Send" button, you should receive a "200 OK" response with API data.



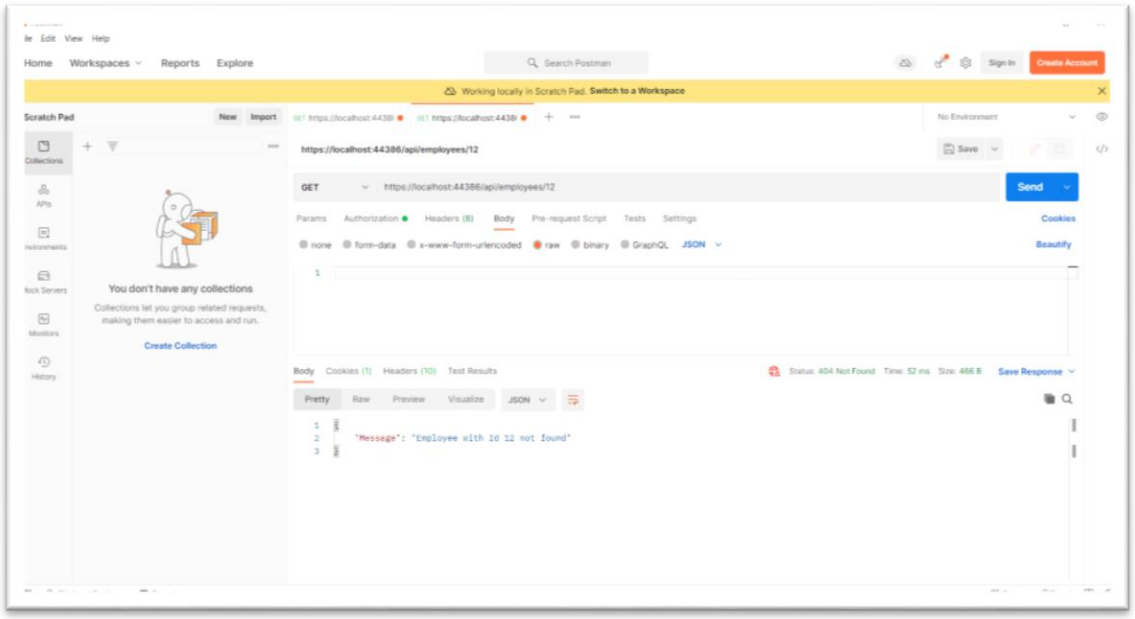
3. To create new data using generated Token with the API, follow these steps:
    1. Open a new request tab by clicking the plus (+) button at the end of the tabs.
    2. Change the http request method to "POST" with the dropdown selector on the left of the URL input field.
    3. In the URL field enter the address to the register route of your local API - https://localhost:44386/api/employees
    4. Select the "Body" tab below the URL field, Select Raw and then select the JSON format. Enter the new user details is body component.
    5. Click the "Send" button, you should receive a "200 OK" response with API data.



4. To update existing data using generated Token with the API, follow these steps:
    1. Open a new request tab by clicking the plus (+) button at the end of the tabs.
    2. Change the http request method to "PUT" with the dropdown selector on the left of the URL input field.
    3. In the URL field enter the address to the register route of your local API - https://localhost:44386/api/employees/12
    4. Select the "Body" tab below the URL field, Select Raw and then select the JSON format. Enter the user details to update records is body component.
    5. Click the "Send" button, you should receive a "200 OK" response with API data.



5. To delete existing data using generated Token with the API, follow these steps:
    1. Open a new request tab by clicking the plus (+) button at the end of the tabs.
    2. Change the http request method to "DELETE" with the dropdown selector on the left of the URL input field.
    3. In the URL field enter the address to the register route of your local API - https://localhost:44386/api/employees/12
    4. Click the "Send" button, you should receive a "200 OK" response with API data.



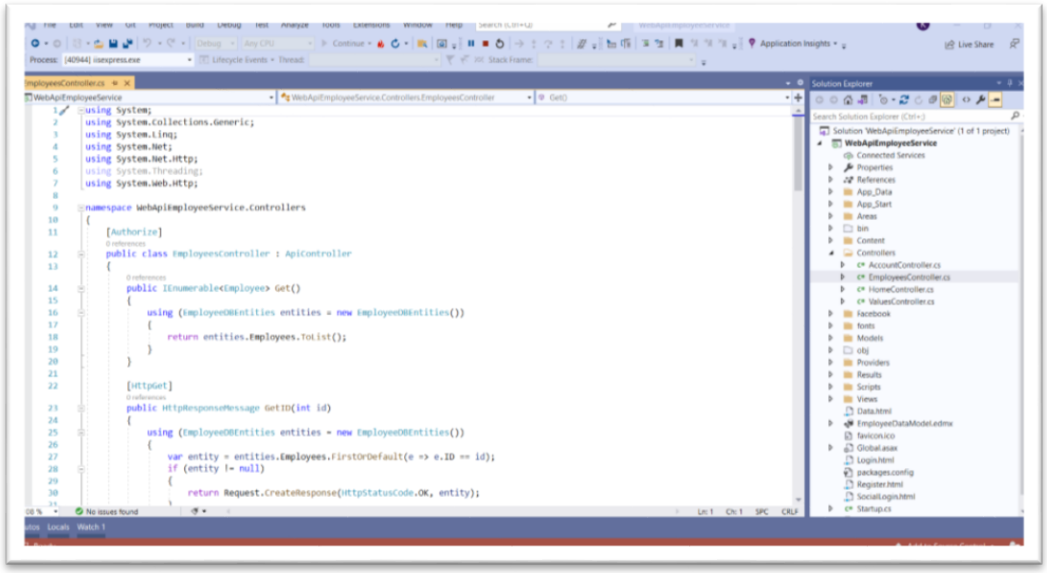
5. Project Information

5.1 Source Code

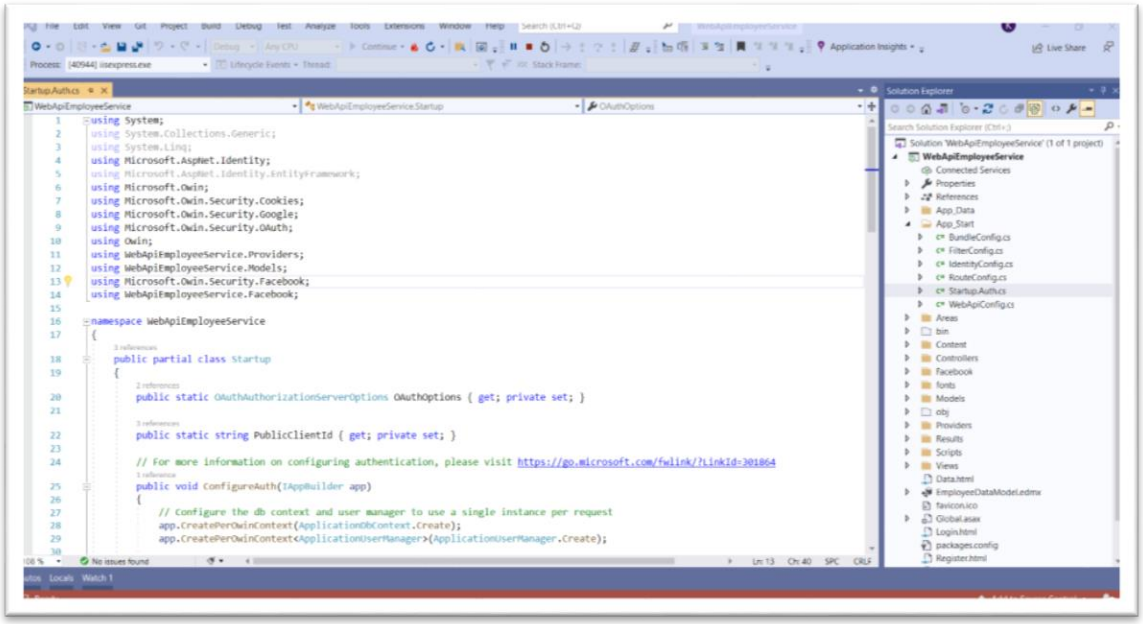
The Entire project Source code in available on Git hub Repository.
https://github.com/KedarSJadhav/MSC_Sem-2_MiniProject

5.2 Project Code Snap Shots

1. Employee Controller



2. Startup.Auth.cs



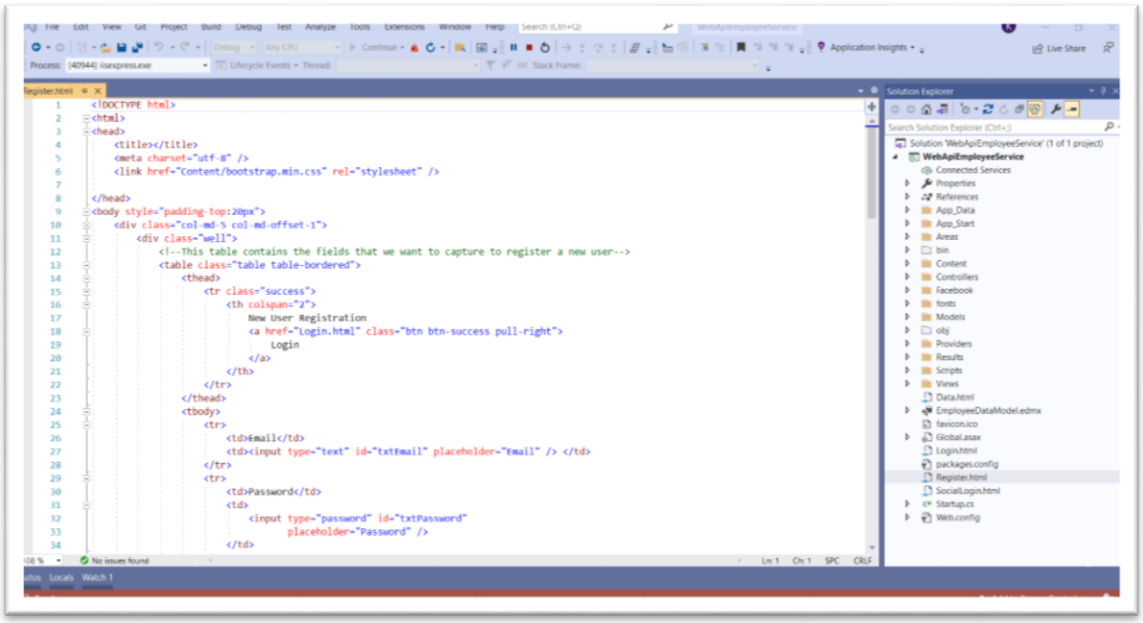
```
using System;
using System.Collections.Generic;
using System.Linq;
using Microsoft.AspNet.Identity;
using Microsoft.AspNet.Identity.EntityFramework;
using Microsoft.Owin;
using Microsoft.Owin.Security.Cookies;
using Microsoft.Owin.Security.Google;
using Microsoft.Owin.Security.OAuth;
using Owin;
using WebApiEmployeeService.Providers;
using WebApiEmployeeService.Models;
using Microsoft.Owin.Security.Facebook;
using WebApiEmployeeService.Facebook;

namespace WebApiEmployeeService
{
    public partial class Startup
    {
        public static OAuthAuthorizationServerOptions OAuthOptions { get; private set; }

        public static string PublicClientId { get; private set; }

        // For more information on configuring authentication, please visit https://go.microsoft.com/fwlink/?LinkId=301864
        public void ConfigureAuth(IAppBuilder app)
        {
            // Configure the db context and user manager to use a single instance per request
            app.CreatePerOwinContext(ApplicationDbContext.Create);
            app.CreatePerOwinContext<ApplicationUserManager>(ApplicationUserManager.Create);
```

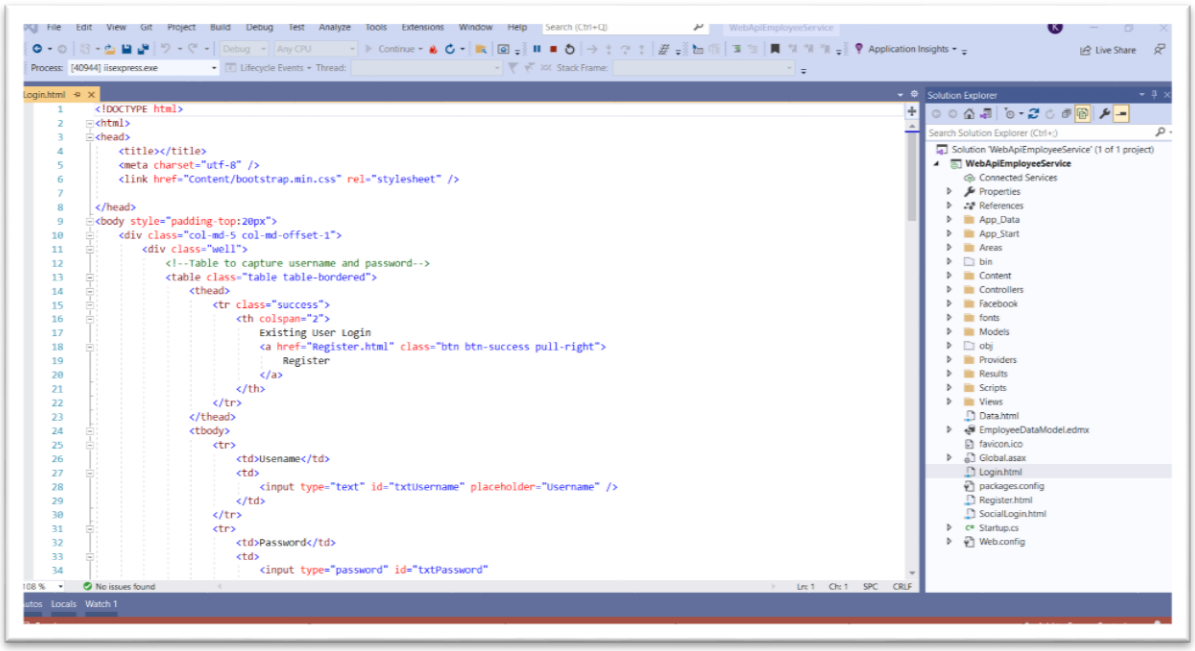
3. Register.html



```
<!DOCTYPE html>
<html>
<head>
    <title></title>
    <meta charset="utf-8" />
    <link href="Content/bootstrap.min.css" rel="stylesheet" />

</head>
<body style="padding-top:20px">
    <div class="col-md-5 col-md-offset-1">
        <div class="well">
            <!--This table contains the fields that we want to capture to register a new user-->
            <table class="table table-bordered">
                <thead>
                    <tr class="success">
                        <th colspan="2">
                            New User Registration
                            <a href="Login.html" class="btn btn-success pull-right">
                                Login
                            </a>
                        </th>
                    </tr>
                </thead>
                <tbody>
                    <tr>
                        <td>Email</td>
                        <td><input type="text" id="txtEmail" placeholder="Email" /> </td>
                    </tr>
                    <tr>
                        <td>Password</td>
                        <td>
                            <input type="password" id="txtPassword"
                                   placeholder="Password" />
                        </td>
```

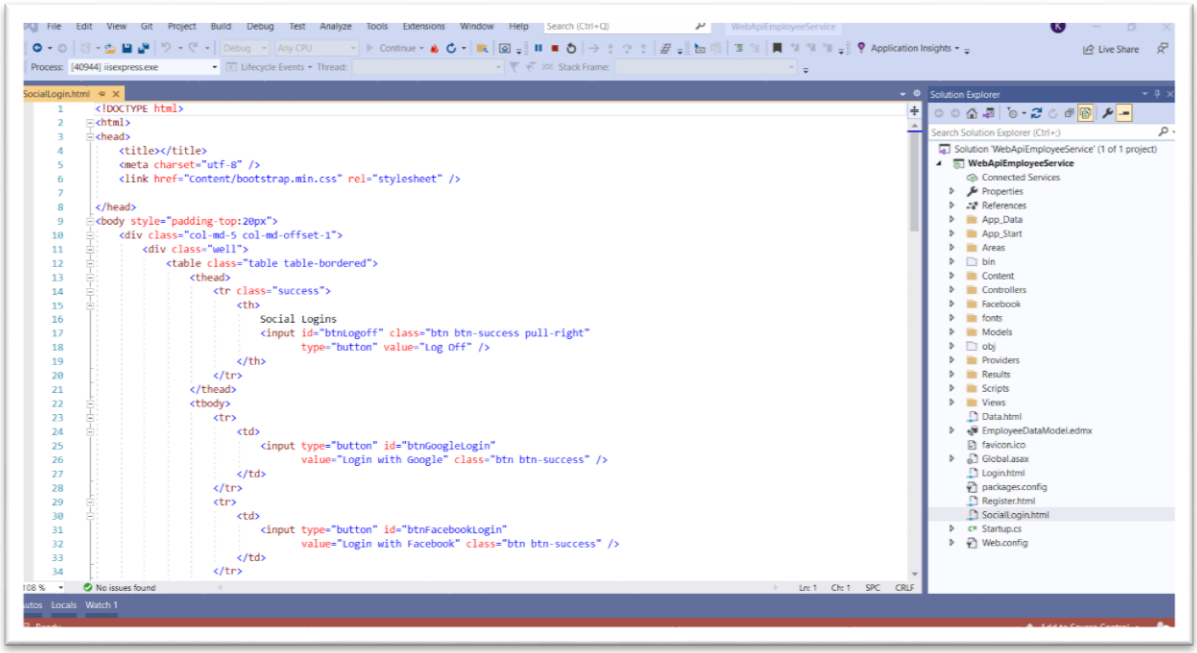
4. Login.html



```
<!DOCTYPE html>
<html>
<head>
    <title></title>
    <meta charset="utf-8" />
    <link href="Content/bootstrap.min.css" rel="stylesheet" />

</head>
<body style="padding-top:20px">
    <div class="col-md-5 col-md-offset-1">
        <div class="well">
            <!--Table to capture username and password-->
            <table class="table table-bordered">
                <thead>
                    <tr class="success">
                    <th colspan="2">
                        Existing User Login
                        <a href="Register.html" class="btn btn-success pull-right">
                            Register
                        </a>
                    </th>
                    </tr>
                </thead>
                <tbody>
                    <tr>
                        <td>Usename</td>
                        <td>
                            <input type="text" id="txtUsername" placeholder="Username" />
                        </td>
                    </tr>
                    <tr>
                        <td>Password</td>
                        <td>
                            <input type="password" id="txtPassword"
```

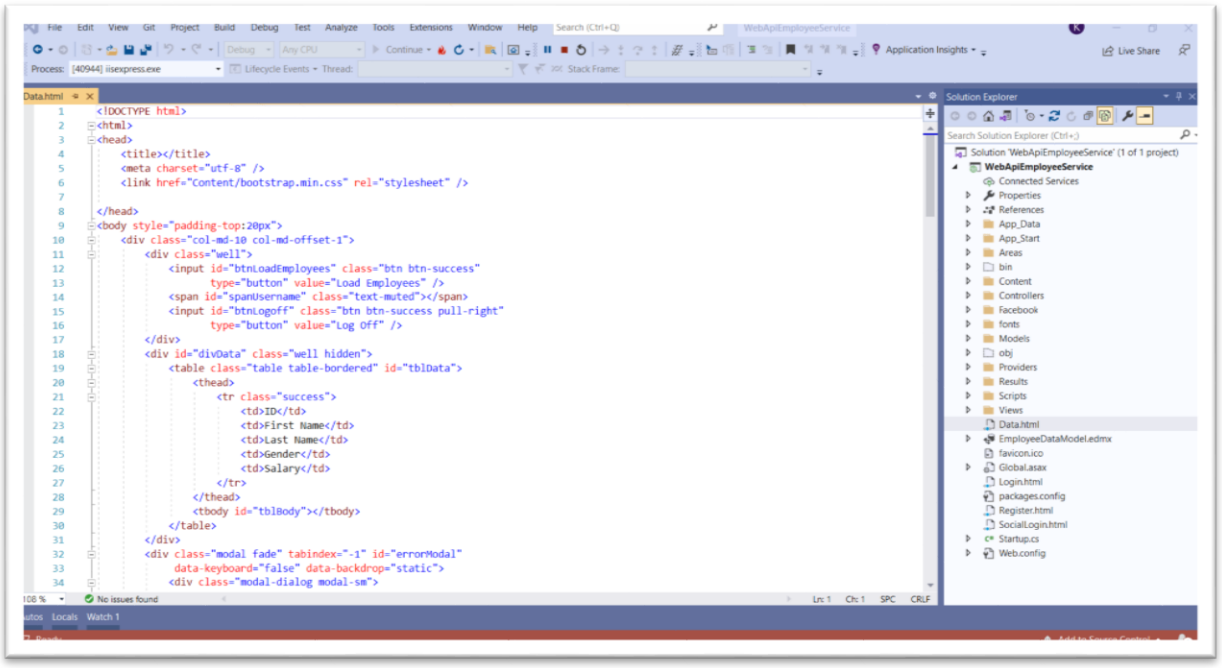
5. SocialLogin.html



```
<!DOCTYPE html>
<html>
<head>
    <title></title>
    <meta charset="utf-8" />
    <link href="Content/bootstrap.min.css" rel="stylesheet" />

</head>
<body style="padding-top:20px">
    <div class="col-md-5 col-md-offset-1">
        <div class="well">
            <table class="table table-bordered">
                <thead>
                    <tr class="success">
                        <th>
                            Social Logins
                            <input id="btnLogoff" class="btn btn-success pull-right"
                                   type="button" value="Log Off" />
                        </th>
                    </tr>
                </thead>
                <tbody>
                    <tr>
                        <td>
                            <input type="button" id="btnGoogleLogin"
                                   value="Login with Google" class="btn btn-success" />
                        </td>
                    </tr>
                    <tr>
                        <td>
                            <input type="button" id="btnFacebookLogin"
                                   value="Login with Facebook" class="btn btn-success" />
                        </td>
                    </tr>
```

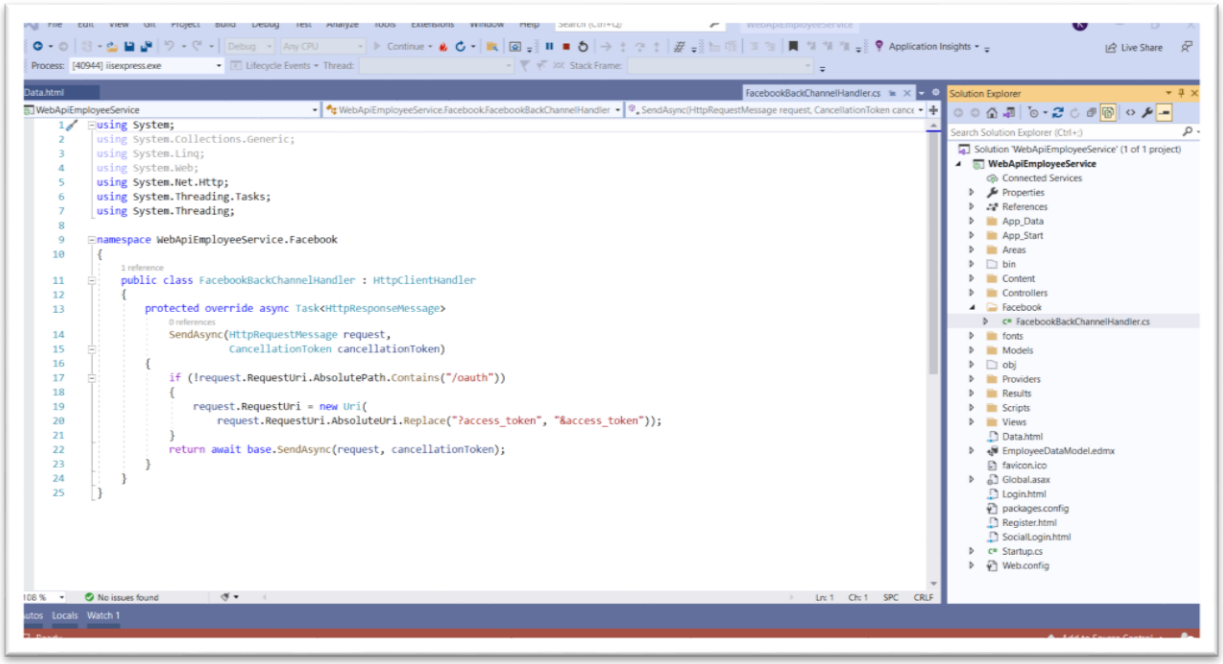
6. Data.html



```
<!DOCTYPE html>
<html>
<head>
    <title></title>
    <meta charset="utf-8" />
    <link href="Content/bootstrap.min.css" rel="stylesheet" />

</head>
<body style="padding-top:20px">
    <div class="col-md-10 col-md-offset-1">
        <div class="well">
            <input id="btnLoadEmployees" class="btn btn-success"
                   type="button" value="Load Employees" />
            <span id="spanUsername" class="text-muted"></span>
            <input id="btnLogoff" class="btn btn-success pull-right"
                   type="button" value="Log Off" />
        </div>
        <div id="divData" class="well hidden">
            <table class="table table-bordered" id="tblData">
                <thead>
                    <tr class="success">
                        <td>ID</td>
                        <td>First Name</td>
                        <td>Last Name</td>
                        <td>Gender</td>
                        <td>Salary</td>
                    </tr>
                </thead>
                <tbody id="tblBody"></tbody>
            </table>
        </div>
        <div class="modal fade" tabindex="-1" id="errorModal"
             data-keyboard="false" data-backdrop="static">
            <div class="modal-dialog modal-sm">
```

7. FacebookBackChannelHandler.cs



```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Net.Http;
using System.Threading.Tasks;
using System.Threading;

namespace WebApiEmployeeService.Facebook
{
    1 reference
    public class FacebookBackChannelHandler : HttpClientHandler
    {
        protected override async Task<HttpResponseMessage>
            0 references
            SendAsync(HttpRequestMessage request,
                      CancellationToken cancellationToken)
        {
            if (!request.RequestUri.AbsolutePath.Contains("/oauth"))
            {
                request.RequestUri = new Uri(
                    request.RequestUri.AbsoluteUri.Replace("?access_token", "&access_token"));
            }
            return await base.SendAsync(request, cancellationToken);
        }
    }
}
```

5.3 Project References

1. Create a web API with ASP.NET
   https://docs.microsoft.com/en-us/aspnet/core/tutorials/first-web-api?view=aspnetcore-6.0&tabs=visual-studio

2. Perform Authentication in Web API
   https://docs.microsoft.com/enus/aspnet/core/security/authentication/identity-api-authorization?view=aspnetcore-6.0

3. JWT token
   https://jwt.io/introduction

4. Web API Google and Facebook Authentication
   https://docs.microsoft.com/en-us/aspnet/web-api/overview/security/external-authentication-services

   https://www.pragimtech.com/courses/asp-net-web-api-tutorial-for-beginners/

6. Project Conclusion

Building a successful API is an art, comprising business analysis, technology architecture, software development, partnership, content writing, developer relations, support, and marketing.
It takes a village to build a good, popular API. In this book, we reviewed the best practices and theory for solid API design; we demonstrated a step-by-step practical use case, and we showed you how to build and maintain a developer ecosystem around an API.

Security –

The information contained within the JSON object can be verified and trusted because it is digitally signed. Although JWTs can also be encrypted to provide secrecy between parties, Auth0-issued JWTs are JSON Web Signatures (JWS), meaning they are signed rather than encrypted. As such, we will focus on signed tokens, which can verify the integrity of the claims contained within them, while encrypted tokens hide those claims from other parties.

In general, JWTs can be signed using a secret (with the HMAC algorithm) or a public/private key pair using RSA or ECDSA (although Auth0 supports only HMAC and RSA). When tokens are signed using public/private key pairs, the signature also certifies that only the party holding the private key is the one that signed it.

Registration is simple and easy using Google authentication. All they have to provide is their social login username and password and the user is registered with our application. This also means one less password to remember. When users don't have to remember multiple usernames and passwords to login to multiple web sites, there will be less failed logins. As you know remembering multiple usernames and passwords is definitely as hassle. Using an external authentication service saves end users from having to create another account for your web application, and also from having to remember another username and password.