

Final Project BDAT 1001

INFORMATION ENCODING
STANDARDS

Solution to Final Group Project on Authentication & Authorization

Group Members

#	Name	ID
1	HABEEB OMOTUNDE	200584861
2	SETH NANA KWAME APPIAH-KUBI	200564282
3	HETVI MANISHBARI BAVARVA	200565677

Manuscript Submitted
on
Date: 30th November 2023

Question:

Create an ASP.NET Core web app with user data protected by authorization

[Create an ASP.NET Core app with user data protected by authorization | Microsoft Docs](https://docs.microsoft.com/en-us/aspnet/core/security/authorization/secure-data/samples?view=aspnetcore-6.0)

Steps taken by group to execute the solution

1. Download the starter app from <https://codeload.github.com/dotnet/AspNetCore.Docs/zip/main> and unzip the AspNetCore.Docs-main.zip file.

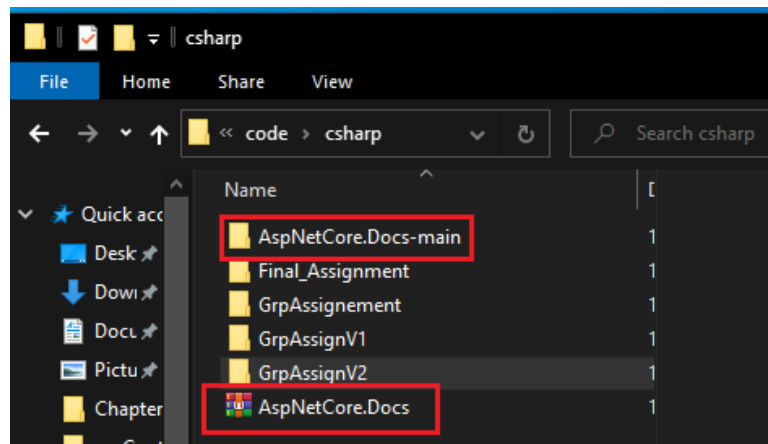


Fig 1. Starter App Download

2. Navigate to the samples directory ".\AspNetCore.Docs-main\AspNetCore.Docs-main\aspnetcore\security\authorization\secure-data\samples" where the starter app is located, copy starter6 folder to the desired directory and rename folder to myContactManager.

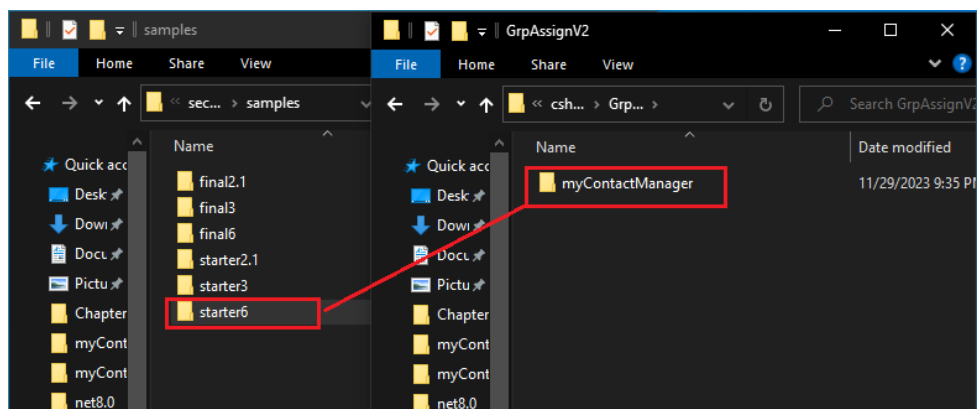


Fig 2. Copy starter6 from samples folder to desired directory & rename

3. Open Visual Studio 2022 and load the Contact Manager solution, “filename.sln”

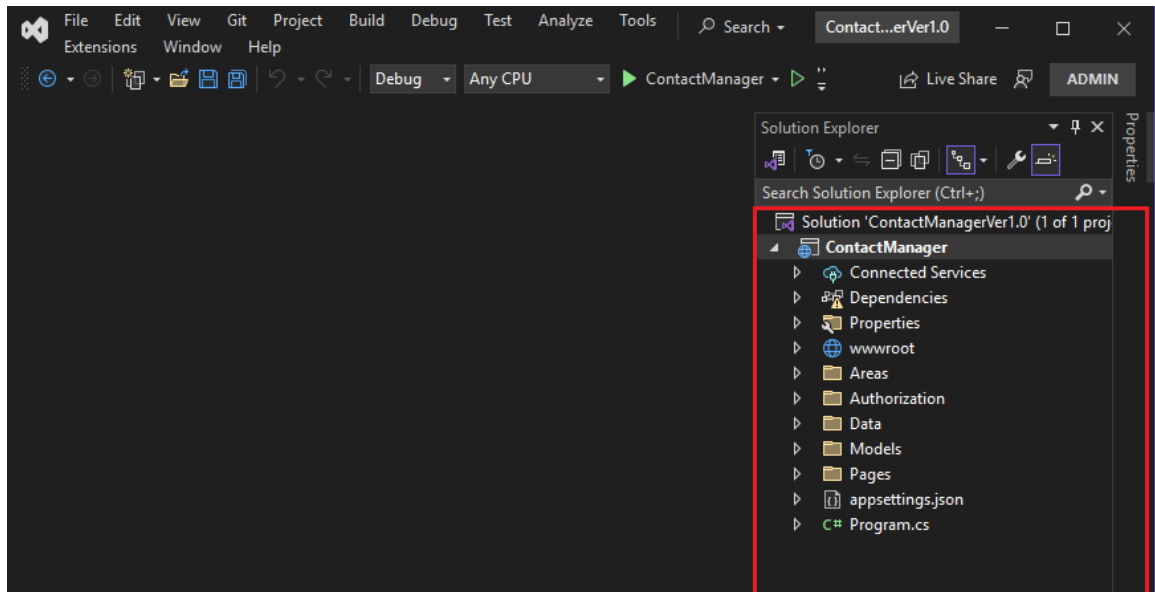


Fig 3. Open the Contactmanager.sln file in Visual Studio 2022

4. Important configurations to setup while following the guide:

- a. Install all recommended packages using NuGet Package manager such as:
 - i. Microsoft.EntityFrameworkCore.SqlServer
 - ii. Microsoft.EntityFrameworkCore.Tools
 - iii. Microsoft.AspNetCore.Identity.EntityFrameworkCore
 - iv. Microsoft.AspNetCore.Authentication.JwtBearer

Note: The packages are pre-installed in the Starter app hence reinstallation is unnecessary

- b. Set the necessary parameters in **appsettings.json** to point to the appropriate SQLServer and provide the database name.

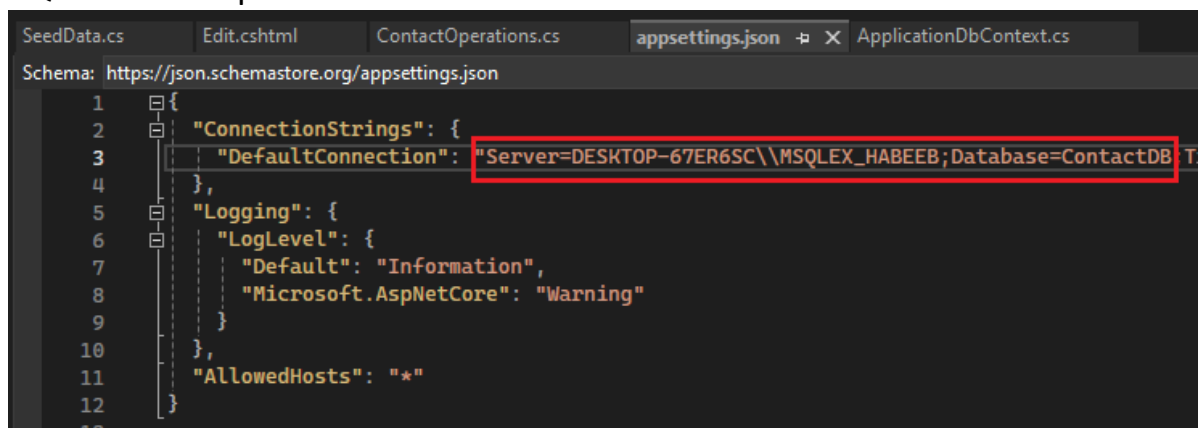
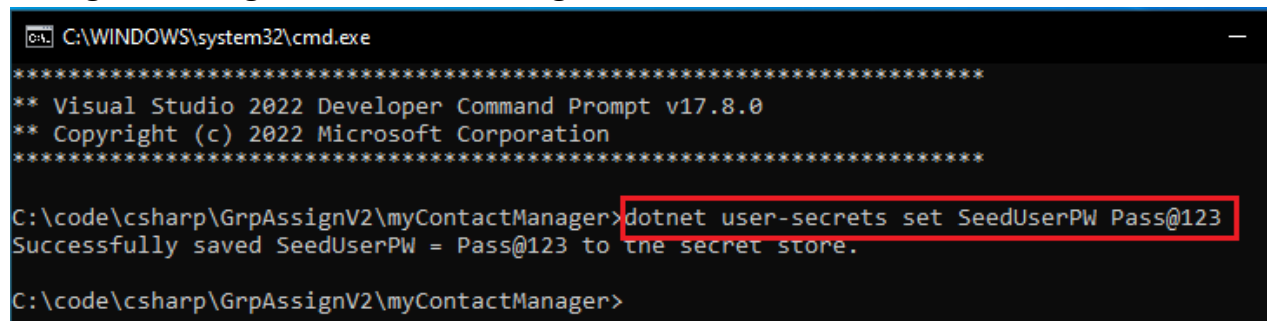


Fig 4. Set DB Connection string and DB name on SQLSERVER

- c. The starter app enforces strict security policies therefore, the Secret parameter is configured using Visual Studio Secret Manager Tool which is accessed using the Developer Command Prompt under “**Tools > command line**” menu in Visual Studio 2022. This contrasts with the usual practice of placing the secret key in plain text which could lead to breach of security and unauthorized access to confidential data.

With the secret manager, lots of parameters like database connection string, database name, server credentials, API keys etc. can be kept confidential while accessible when needed.

In the Contact manager App, the secret key is stored as “**SeedUserPW**”. This is configured using the command in Fig 5 below:



```
C:\WINDOWS\system32\cmd.exe

** Visual Studio 2022 Developer Command Prompt v17.8.0
** Copyright (c) 2022 Microsoft Corporation
*****

C:\code\csharp\GrpAssignV2\myContactManager>dotnet user-secrets set SeedUserPW Pass@123
Successfully saved SeedUserPW = Pass@123 to the secret store.

C:\code\csharp\GrpAssignV2\myContactManager>
```

Fig 5. Setting the secret key using the Secret Manager Tool

- d. Set up database using the Package Manager console.

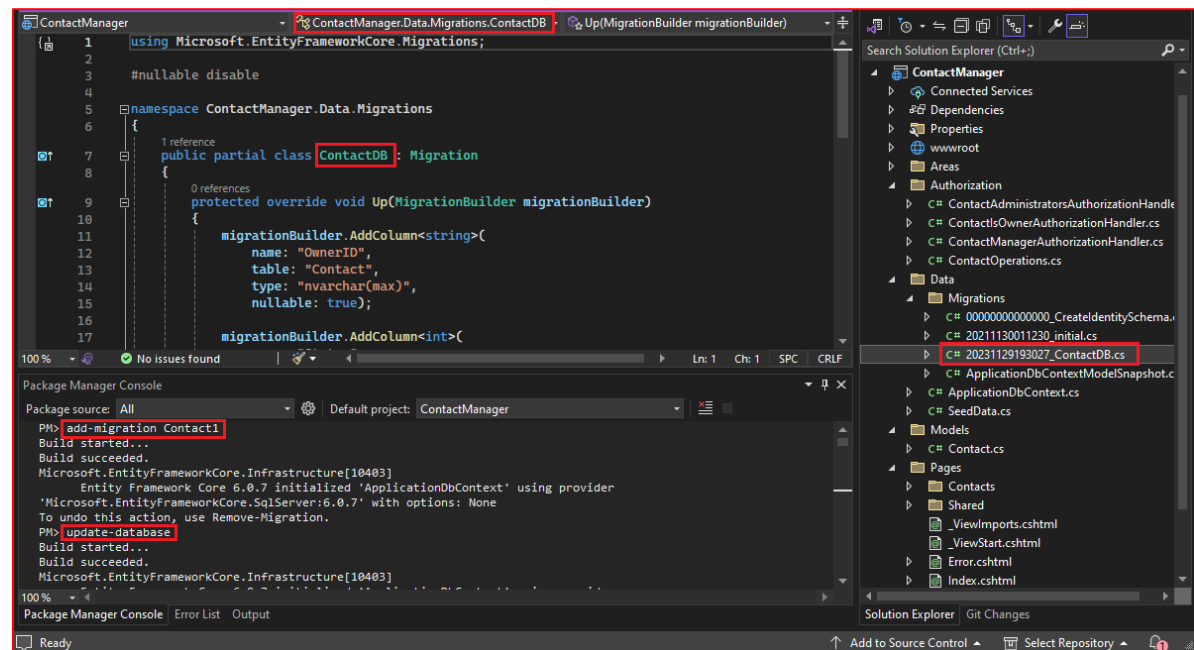


Fig 6. Generating DB migration scripts and DB Creation

- e. The database, ContactDB, is created after running the commands in Fig 6 and this can be seen in SQL Server Management Studio in Fig 7.

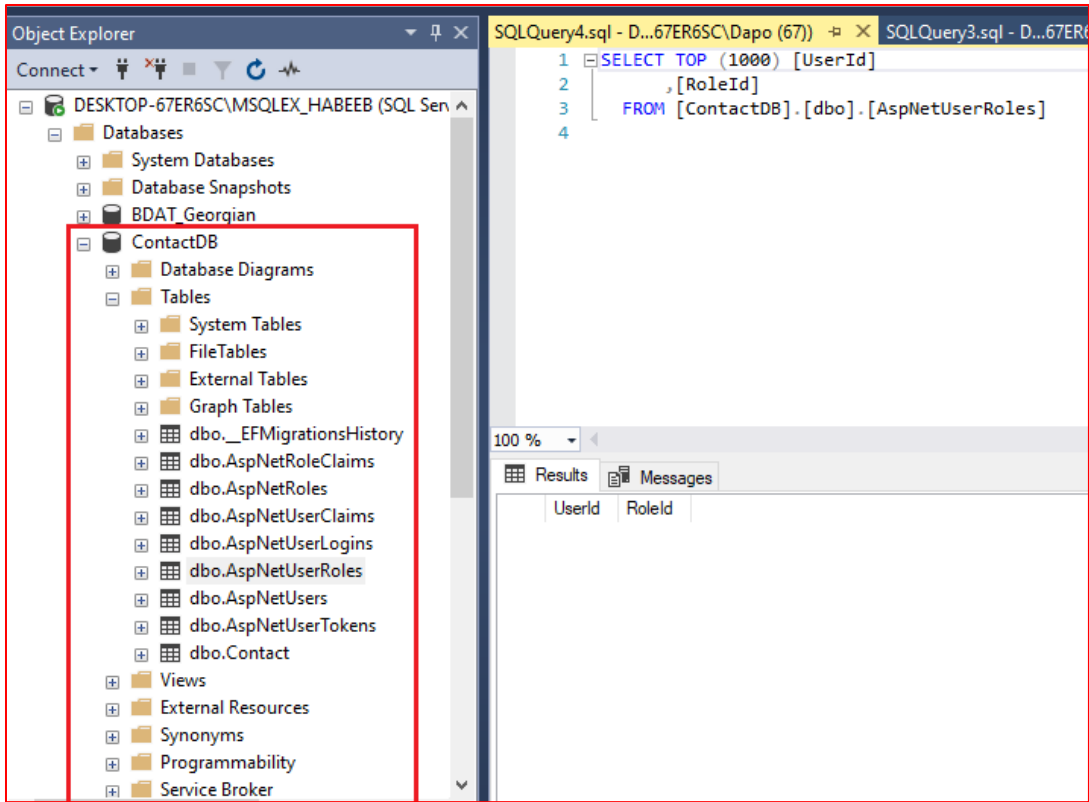


Fig 7. Contact DB created on SQL Server as shown in SMSS

- f. To enforce authorization on the contact data saved in the Contact DB, each contact record is associated with an owner (UserID on AspNetUsers table) by modifying the Contact model, i.e.

```
// user ID from AspNetUser table.
public string? OwnerID { get; set; }
```

Each contact has a status which could either be Submitted, Approved or Rejected.

- g. Next, enable Role-Based Authorization by adding Role services to each identity using the AddRoles() method of the IdentityBuilder class. This is because when an identity is created it may belong to one or more roles. The starter app has 3 roles namely:

	Role	Purpose	Handler
1	User	Registered users can view all the approved data and can edit/delete their own data	ContactIsOwnerAuthorizationHandler
2	Manager	Managers can approve or reject contact data. Only approved contacts are visible to users.	ContactManagerAuthorizationHandler
3	Admin	Administrators can approve/reject and edit/delete any data	ContactAdministratorsAuthorizationHandler

Table 1: Roles and Handlers in startup app.

- h. Set the fallback authorization policy to require users to be authenticated, therefore no anonymous access except for pages or controllers with authorization attribute [AllowAnonymous]. This will make sure any user who intends to access the app must be authenticated.

```
builder.Services.AddAuthorization(options =>
{
    options.FallbackPolicy = new AuthorizationPolicyBuilder()
        .RequireAuthenticatedUser() //ensures no anonymous access
        .Build();
});
```

Fig 8. Adding Authentication Service to starter app

- i. Configure the test accounts in the SeedData class. This class creates two accounts using it's **Initialize** method and also assigns each user to a role by calling the respective methods.

```
var adminID = await EnsureUser(serviceProvider, testUserPw, "admin@contoso.com");
await EnsureRole(serviceProvider, adminID, ContactManager.Authorization.Constants.ContactAdministratorsRole);

// allowed user can create and edit contacts that they create
var managerID = await EnsureUser(serviceProvider, testUserPw, "manager@contoso.com");
await EnsureRole(serviceProvider, managerID, ContactManager.Authorization.Constants.ContactManagersRole);
```

Fig 8. Create users and assign roles

- j. To create the accounts correctly, a strong password must be set in step 4c else the exception below will be thrown:

Unhandled exception. System.Exception: The testUserPw password was probably not strong enough!

- k. Create a new folder in the project and rename it to Authorization. This folder will contain all handlers in Table 1. Follow all instruction in the guide to create the handlers and ensure their namespaces are accurate. Then Register the authorization handlers in **Program.cs** with the service collection so they're available to the **ContactsController** through dependency injection.

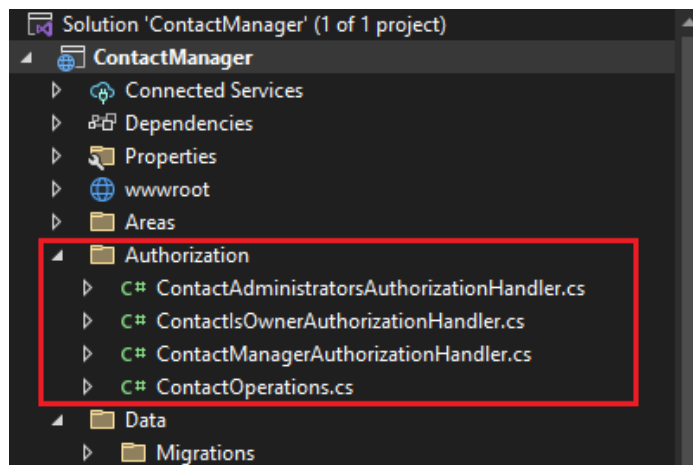


Fig 8. Authorization and Handlers

```
// Authorization handlers.
```

```
builder.Services.AddScoped<IAuthorizationHandler,  
    ContactIsOwnerAuthorizationHandler>();
```

```
builder.Services.AddSingleton<IAuthorizationHandler,  
    ContactAdministratorsAuthorizationHandler>();
```

```
builder.Services.AddSingleton<IAuthorizationHandler,  
    ContactManagerAuthorizationHandler>();
```

I. Execute the following instructions to handle CRUD operations:

#	Operation	Description
1	Update the CreateModel	Call the authorization handler to verify the user has permission to create contacts
2	Update the IndexModel :	Ensure only approved contacts are shown to general users
3	Update the EditModel :	Add an authorization handler to verify the user owns the contact before the user is allowed to make changes.
4	Update the DeleteModel	Update the delete page model to use the authorization handler to verify the user has delete permission on the contact.

m. Update Views as provided by the guide

5. Build & run the completed application.

- Ensure a strong password is set with the Secret Manager tool
- Before starting the application, ensure the Microsoft SQL server is running.
- Note the two user records created and inserted into the `dbo.AspNetUsers` table on ContactDB. These are the manager and administrator users.

The screenshot shows the SQL Server Enterprise Manager interface. On the left, the Object Explorer displays the database structure for 'DESKTOP-67ER65C\MSSQL\HABEEB (SQL Sen ^)'. The 'ContactDB' database is expanded, showing 'Tables' and 'dbo.AspNetUsers'. The 'dbo.AspNetUsers' table is highlighted with a red box. On the right, the SQL Query window shows a query: `SELECT TOP (1000) [Id], [UserName], [NormalizedUserName], [Email], [NormalizedEmail], [EmailConfirmed], [PasswordHash], [SecurityStamp], [ConcurrencyStamp], [PhoneNumber], [PhoneNumberConfirmed], [TwoFactorEnabled], [LockoutEnd], [LockoutEnabled], [AccessFailedCount] FROM [ContactDB].[dbo].[AspNetUsers]`. The 'Results' tab shows two rows of data:

Id	UserName	NormalizedUserName	Email	NormalizedEmail
1	manager@contoso.com	MANAGER@CONTOSO.COM	NULL	NULL
2	admin@contoso.com	ADMIN@CONTOSO.COM	NULL	NULL

Fig 9. Users inserted into table `AspNetUsers` after running the starter App

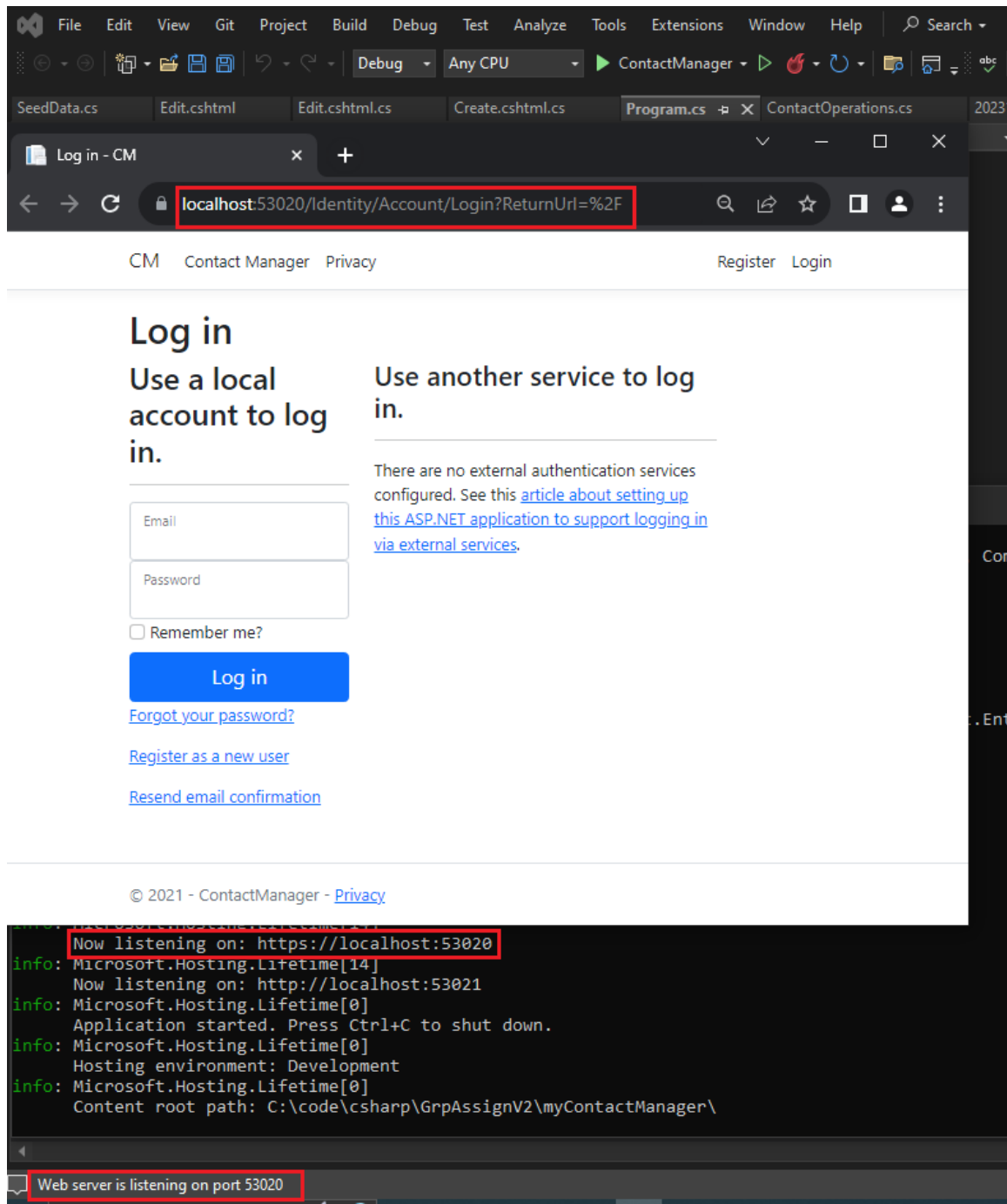


Fig 10. ContactApp running on webserver port 53020

6. Login with either of these users using the **SeedUserPW** configured with the secret tools manager. The Figure below shows a login with manager.
A **manager** can approve/reject contact data. The Details view shows Approve and Reject buttons. Note that managers can not edit or delete contact data.

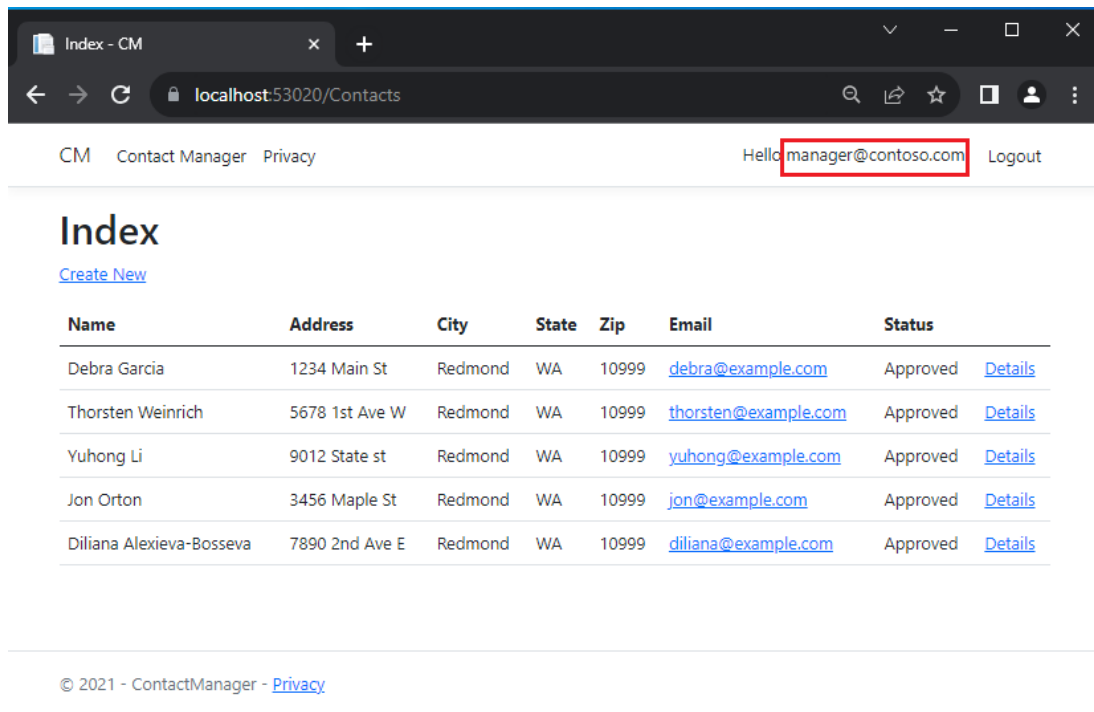


Fig 11. Manager login

7. In the next scenario,
 - a. A new user (fromhabib2u@georgiancollege.ca) will be registered and record will be inserted into the database.

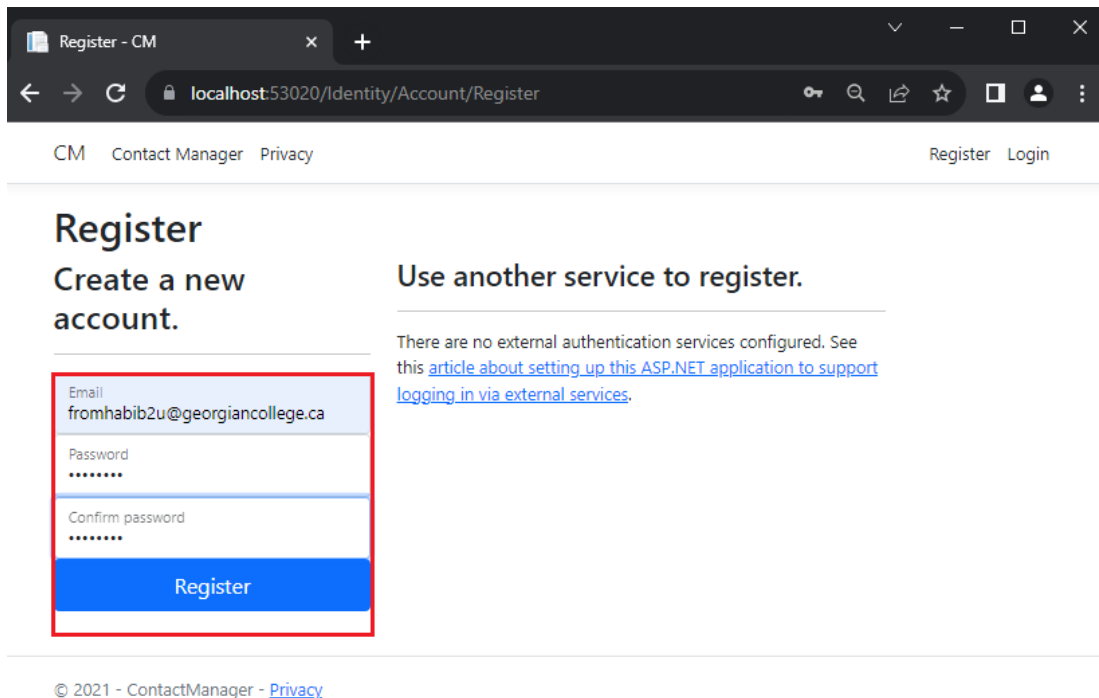


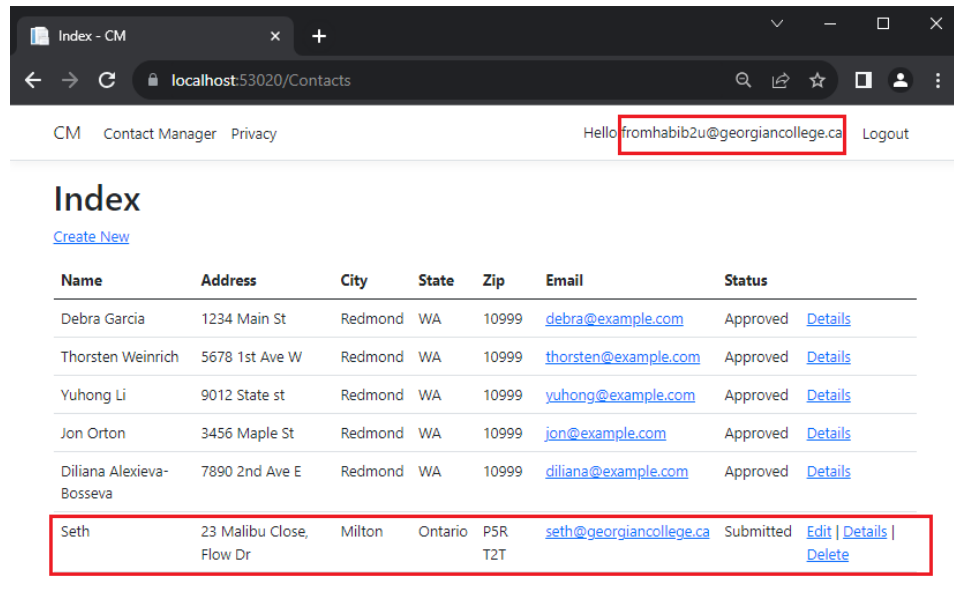
Fig 12.

New User Registration

Results			
Messages			
	Id	UserName	NormalizedUserName
1	3cfd8115-30ca-403e-86ae-52e81a9eb519	fromhabib2u@georgiancollege.ca	FROMHABIB2U@GEORGIANCOLLEGE.CA

Fig 13. New record inserted into the Contact DB

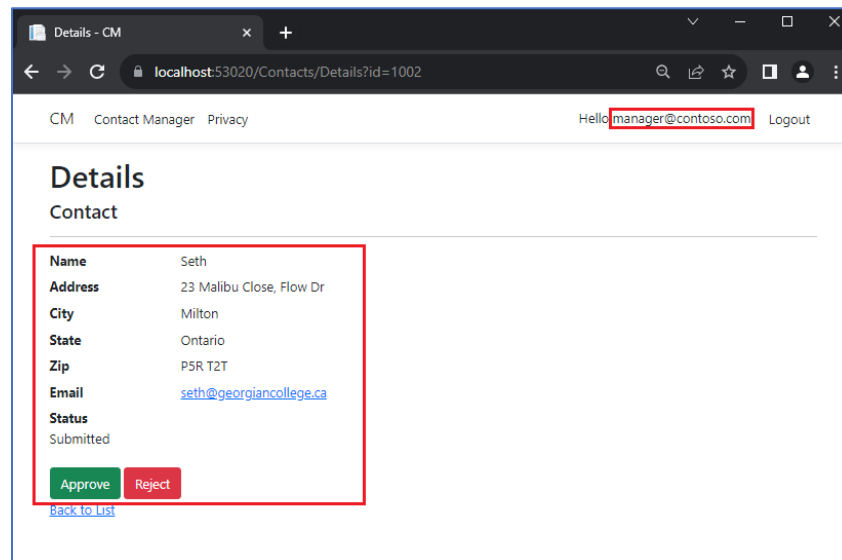
- b. The user will create a contact ,as highlighted in Fig 14, which will appear on manager’s console requesting approval or rejection. Also, only the newly created contact displays edit, details and delete links.



Name	Address	City	State	Zip	Email	Status
Debra Garcia	1234 Main St	Redmond	WA	10999	debra@example.com	Approved Details
Thorsten Weinrich	5678 1st Ave W	Redmond	WA	10999	thorsten@example.com	Approved Details
Yuhong Li	9012 State st	Redmond	WA	10999	yuhong@example.com	Approved Details
Jon Orton	3456 Maple St	Redmond	WA	10999	jon@example.com	Approved Details
Diliana Alexieva-Bosseva	7890 2nd Ave E	Redmond	WA	10999	diliana@example.com	Approved Details
Seth	23 Malibu Close, Flow Dr	Milton	Ontario	P5R T2T	seth@georgiancollege.ca	Submitted Edit Details Delete

Fig 14. New user creates a contact.

- c. Manager can view detail of submitted contact & either approve or reject it. In this case, contact is approved.



Name	Seth
Address	23 Malibu Close, Flow Dr
City	Milton
State	Ontario
Zip	P5R T2T
Email	seth@georgiancollege.ca
Status	Submitted

[Back to List](#)

[Approve](#) [Reject](#)

Fig 15. New contact approved by Manager

- d. When the new user views it as seen in Fig 16, status will change from Submitted > Approved.

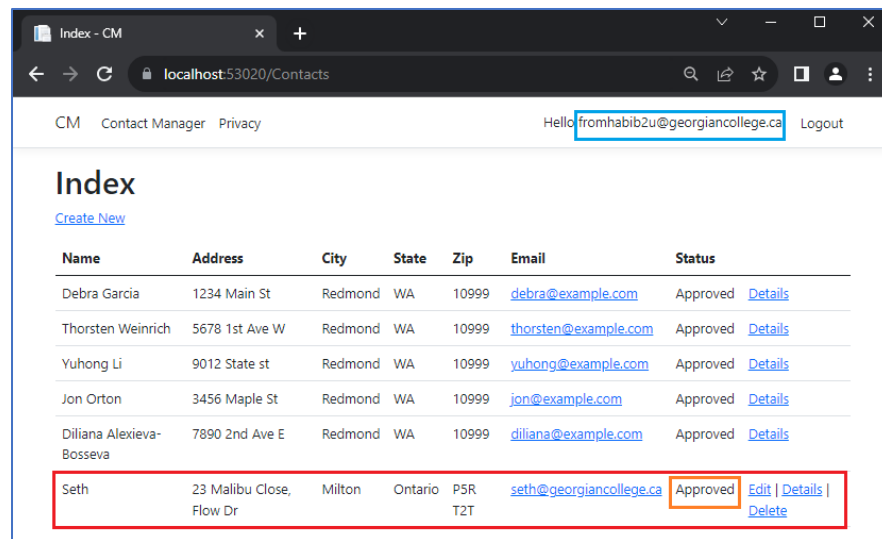


Fig 16. New contact approved by Manager

- e. Admin will be able to review approval, view details, edit or delete contact to demonstrate their different access rights. In this case he decides to override the manager's approval and reject the contact.

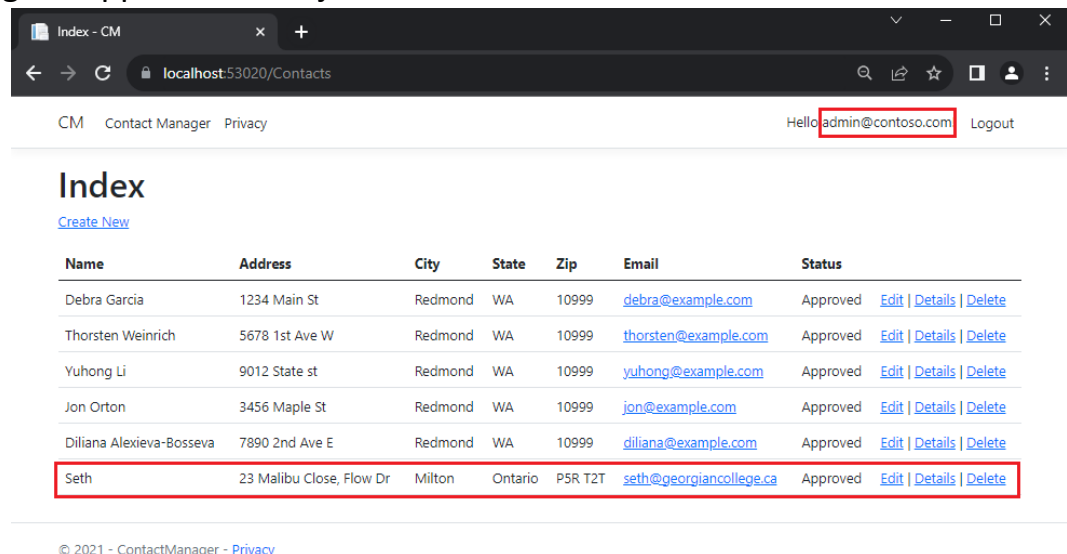


Fig 17. Admin reviews approved contact

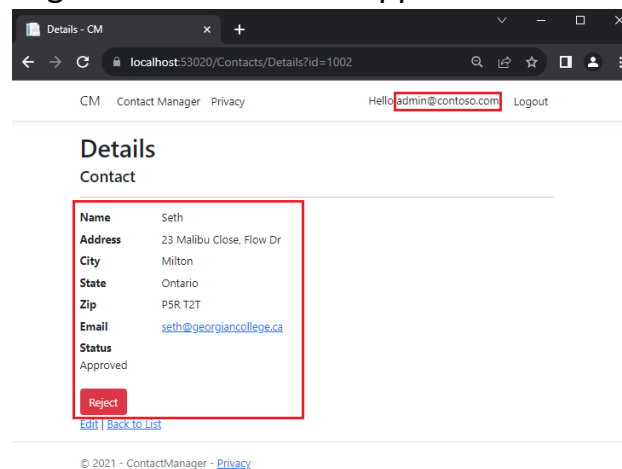


Fig 18. Contact rejected by admin

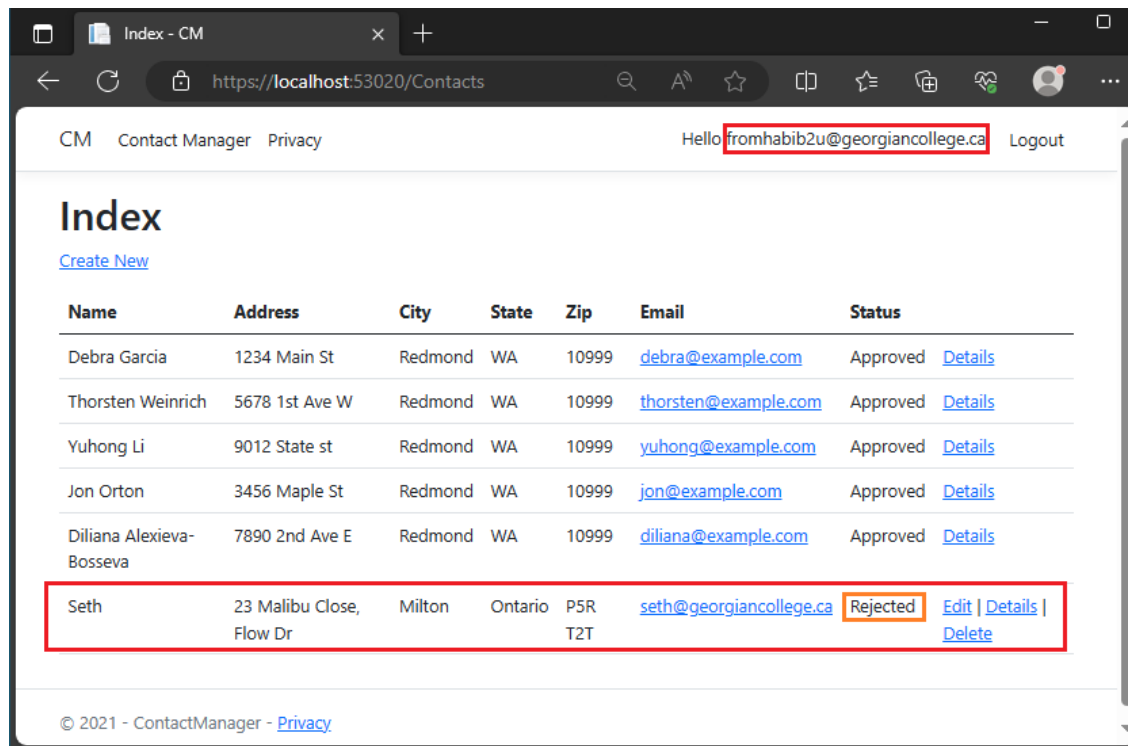


Fig 19. New User views updated status of contact

8. In summary, the sample Contact App has been used in demonstrating Authentication and authorization in securing user data and enforcing confidentiality. As a recap, the starter app has 3 security groups namely, a registered user, manager and admin. Their respective access right are summarized below:
- Registered users can view all the approved data and can edit/delete their own data.
 - Managers can approve or reject contact data. Only approved contacts are visible to users.
 - Administrators can approve/reject and edit/delete any data.

Finally, It's important to mention that the group would have loved to implement a feature named **delegate** where a new or existing user in the ContactDB can be promoted to an administrator or manager. This role can only be executed by the Administrator.