CLDV6212 POE PART 3

Miguel Almeida ST10025374

Part A

Azure Components

Component	Technology Choice	Hosting Model
Azure Function (HTTP Trigger)	Compute	Serverless
Azure Storage Queue	Data Storage	PaaS
Azure SQL Database	Data Storage	PaaS
Azure Function (Queue trigger)	Compute	PaaS

Part B

Motivation for the use of Azure Blob Storage

Given the scenario I decided to change the Azure SQL Database to Azure Blob storage. The first design using Azure SQL Database worked well for structured data but not so well for unstructured data, because the data that is stored must be compatible with the columns specified in the database table. Azure Blob Storage is ideal for storing unstructured data. As an important need from Aweh Productions, this change makes it easier to work with different data formats. Unstructured data is exactly what Azure Blob Storage is made to handle. It doesn't force any structure or data model on you, so processing data is faster and there is less task to do.

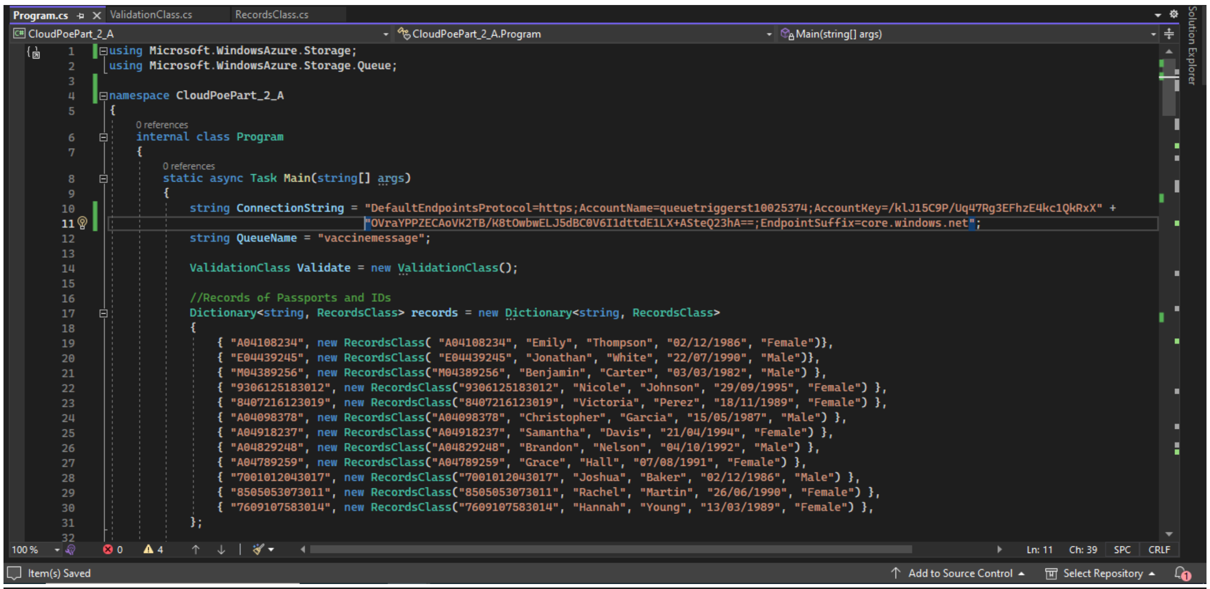
Part C

Console code

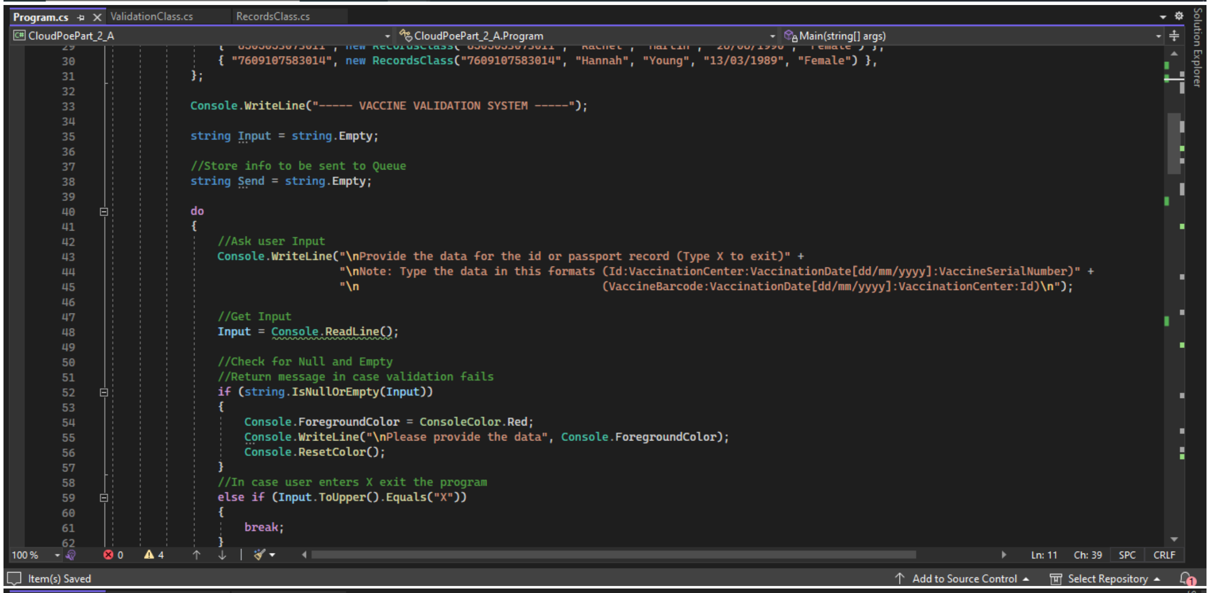
```
using Microsoft.WindowsAzure.Storage;

using Microsoft.WindowsAzure.Storage.Queue;


namespace CloudPoePart_2_A

{

  internal class Program

  {

    static async Task Main(string[] args)

    {

      string ConnectionString =
"DefaultEndpointsProtocol=https;AccountName=queuetriggerst10025374;AccountKey=/klJ15C9P/Uq
47Rg3EFhzE4kc1QkRxXOVraYPPZECAoVK2TB/K8tOwbwELJ5dBC0V6I1dttdE1LX+ASteQ23hA==;Endpoi
ntSuffix=core.windows.net";

      string QueueName = "vaccinemessage";


      ValidationClass Validate = new ValidationClass();


      //Records of Passports and IDs
```

```
        Dictionary<string, RecordsClass> records = new Dictionary<string, RecordsClass>
        {
            { "A04108234", new RecordsClass( "A04108234", "Emily", "Thompson", "02/12/1986",
"Female")},
            { "E04439245", new RecordsClass( "E04439245", "Jonathan", "White", "22/07/1990",
"Male")},
            { "M04389256", new RecordsClass("M04389256", "Benjamin", "Carter", "03/03/1982",
"Male") },
            { "9306125183012", new RecordsClass("9306125183012", "Nicole", "Johnson",
"29/09/1995", "Female") },
            { "8407216123019", new RecordsClass("8407216123019", "Victoria", "Perez",
"18/11/1989", "Female") },
            { "A04098378", new RecordsClass("A04098378", "Christopher", "Garcia", "15/05/1987",
"Male") },
            { "A04918237", new RecordsClass("A04918237", "Samantha", "Davis", "21/04/1994",
"Female") },
            { "A04829248", new RecordsClass("A04829248", "Brandon", "Nelson", "04/10/1992",
"Male") },
            { "A04789259", new RecordsClass("A04789259", "Grace", "Hall", "07/08/1991", "Female") },
            { "7001012043017", new RecordsClass("7001012043017", "Joshua", "Baker", "02/12/1986",
"Male") },
            { "8505053073011", new RecordsClass("8505053073011", "Rachel", "Martin",
"26/06/1990", "Female") },
            { "7609107583014", new RecordsClass("7609107583014", "Hannah", "Young",
"13/03/1989", "Female") },
        };


        Console.WriteLine("----- VACCINE VALIDATION SYSTEM -----");


        string Input = string.Empty;


        //Store info to be sent to Queue
        string Send = string.Empty;


        do
        {
            //Ask user Input
            Console.WriteLine("\nProvide the data for the id or passport record (Type X to exit)" +
                            "\nNote: Type the data in this formats
(Id:VaccinationCenter:VaccinationDate[dd/mm/yyyy]:VaccineSerialNumber)" +
                            "\n
(VaccineBarcode:VaccinationDate[dd/mm/yyyy]:VaccinationCenter:Id)\n");

            //Get Input
            Input = Console.ReadLine();

            //Check for Null and Empty
            //Return message in case validation fails
            if (string.IsNullOrEmpty(Input))
            {
                Console.ForegroundColor = ConsoleColor.Red;
                Console.WriteLine("\nPlease provide the data", Console.ForegroundColor);
                Console.ResetColor();
            }
            //In case user enters X exit the program
            else if (Input.ToUpper().Equals("X"))
            {
                break;
            }
            else if (!string.IsNullOrEmpty(Input))
            {
                //Check if Input format is correct. If not correct loop again
                if (Validate.CheckUserInputSerialNumber(Input).Equals(true) ||
Validate.CheckUserInputBarCode(Input).Equals(true))
                {
                    string Id = string.Empty;
                    string VaccinationCenter = string.Empty;
                    string VaccinationDate = string.Empty;
                    string VaccineSerialNumber = string.Empty;
                    string VaccineBarCode = string.Empty;

                    //Check if user used barcode or serial number
                    bool choice = Validate.GetUserFormatChoice(Input);

                    //For Serial Number
                    if (choice.Equals(true))
                    {
                        //Split Input
                        string[] parts = Input.Split(':');

                        // Assigning parts to variables
                        Id = parts[0];
                        VaccinationCenter = parts[1];
                        VaccinationDate = parts[2];
                        VaccineSerialNumber = parts[3];

                        // Check if the entered ID or Passport Number exists in the dictionary and get the
corresponding value
                        if (records.TryGetValue(Id, out RecordsClass matchedRecord))
                        {
                            if (matchedRecord.Id.Equals(Id))
                            {
                                //Display full info to user including added data
                                Console.ForegroundColor = ConsoleColor.Green;
                                Console.WriteLine($"\nRecord:                 {Id}" +
                                            $"\nVacination serial number:  {VaccineSerialNumber}" +
                                            $"\nFirst name:            {matchedRecord.FirstName} " +
                                            $"\nSurname:               {matchedRecord.LastName}" +
                                    $"\nDate of birth:          {matchedRecord.DateOfBirth}" +
                                    $"\nGender:                 {matchedRecord.Gender}" +
                                    $"\nVaccination center:     {VaccinationCenter}" +
                                    $"\nVaccination date:       {VaccinationDate}",
Console.ForegroundColor);
                            Console.ResetColor();

                            Send = $"{Id}:{VaccinationCenter}:{VaccinationDate}:{VaccineSerialNumber}";

                            //Connect to the Azure Storage Queue
                            CloudStorageAccount storageAccount =
CloudStorageAccount.Parse(ConnectionString);
                            CloudQueueClient queueClient = storageAccount.CreateCloudQueueClient();
                            CloudQueue queue = queueClient.GetQueueReference(QueueName);

                            // Ensure the queue exists
                            await queue.CreateIfNotExistsAsync();

                            // Save to Azure Storage Queue
                            CloudQueueMessage message = new CloudQueueMessage(Send);
                            await queue.AddMessageAsync(message);

                            Console.WriteLine("\nData saved to Azure Queue.");
                        }
                    }
                    else
                    {
                        //Check if Id or Passport format corresponds to the format of SA Passport and ID
                        if (Validate.CheckValidPassport(Id).Equals(true) ||
Validate.CheckValidID(Id).Equals(true))
                        {
                            Console.ForegroundColor = ConsoleColor.Green;
                            Console.WriteLine($"\nRecord: {Id}   Status: Not vaccinated", 
Console.ForegroundColor);

                            Console.ResetColor();

                        }

                        //In case Id or passport format is not valid display message

                        else

                        {

                            Console.ForegroundColor = ConsoleColor.Red;

                            Console.WriteLine("\nInvalid Passport or Id number format entered", 
Console.ForegroundColor);

                            Console.ResetColor();

                        }

                    }

                }


                // For BarCode

                else

                {

                    //Split Input

                    string[] parts2 = Input.Split(':');


                    VaccineBarCode = parts2[0];

                    VaccinationDate = parts2[1];

                    VaccinationCenter = parts2[2];

                    Id = parts2[3];


                    // Check if the entered ID or Passport Number exists in the dictionary and get the 
corresponding value

                    if (records.TryGetValue(Id, out RecordsClass matchedRecord))

                    {

                        if (matchedRecord.Id.Equals(Id))

                        {
                            //Display full info to user including added data
                            Console.ForegroundColor = ConsoleColor.Green;
                            Console.WriteLine($"\nRecord:                  {Id}" +
                                      $"\nVaccine barcode:          {VaccineBarCode}" +
                                      $"\nFirst name:               {matchedRecord.FirstName} " +
                                      $"\nSurname:                  {matchedRecord.LastName}" +
                                      $"\nDate of birth:            {matchedRecord.DateOfBirth}" +
                                      $"\nGender:                   {matchedRecord.Gender}" +
                                      $"\nVaccination center:       {VaccinationCenter}" +
                                      $"\nVaccination date:         {VaccinationDate}",
Console.ForegroundColor);
                            Console.ResetColor();

                            Send = $"{VaccineBarCode}:{VaccinationDate}:{VaccinationCenter}:{Id}";

                            //Connect to the Azure Storage Queue
                            CloudStorageAccount storageAccount =
CloudStorageAccount.Parse(ConnectionString);
                            CloudQueueClient queueClient = storageAccount.CreateCloudQueueClient();
                            CloudQueue queue = queueClient.GetQueueReference(QueueName);

                            // Ensure the queue exists
                            await queue.CreateIfNotExistsAsync();

                            // Save to Azure Storage Queue
                            CloudQueueMessage message = new CloudQueueMessage(Send);
                            await queue.AddMessageAsync(message);

                            Console.WriteLine("\nData saved to Azure Queue.");
                        }
                    }
                    else
                    {
                      //Check if Id or Passport format corresponds to the format of SA Passport and ID
                      if (Validate.CheckValidPassport(Id).Equals(true) ||
Validate.CheckValidID(Id).Equals(true))
                      {
                        Console.ForegroundColor = ConsoleColor.Green;
                        Console.WriteLine($"\nRecord: {Id}   Status: Not vaccinated",
Console.ForegroundColor);
                        Console.ResetColor();
                      }
                      //In case Id or passport format is not valid display message
                      else
                      {
                        Console.ForegroundColor = ConsoleColor.Red;
                        Console.WriteLine("\nInvalid Passport or Id number format entered",
Console.ForegroundColor);
                        Console.ResetColor();
                      }
                    }
                  }
                }
                //In case user enters incorrect data or vaccine serial number format message will be
displayed
                else
                {
                  Console.ForegroundColor = ConsoleColor.Red;
                  Console.WriteLine("\nInvalid data entered", Console.ForegroundColor);
                  Console.ResetColor();
                }
              }
            }while(!Input.ToUpper().Equals("X"));
            Console.WriteLine("\nSystem will be terminated...");

            Console.ReadLine();

        }

    }

}

//-------------------------------------------------------< END >----------------------------------------------------------//
```

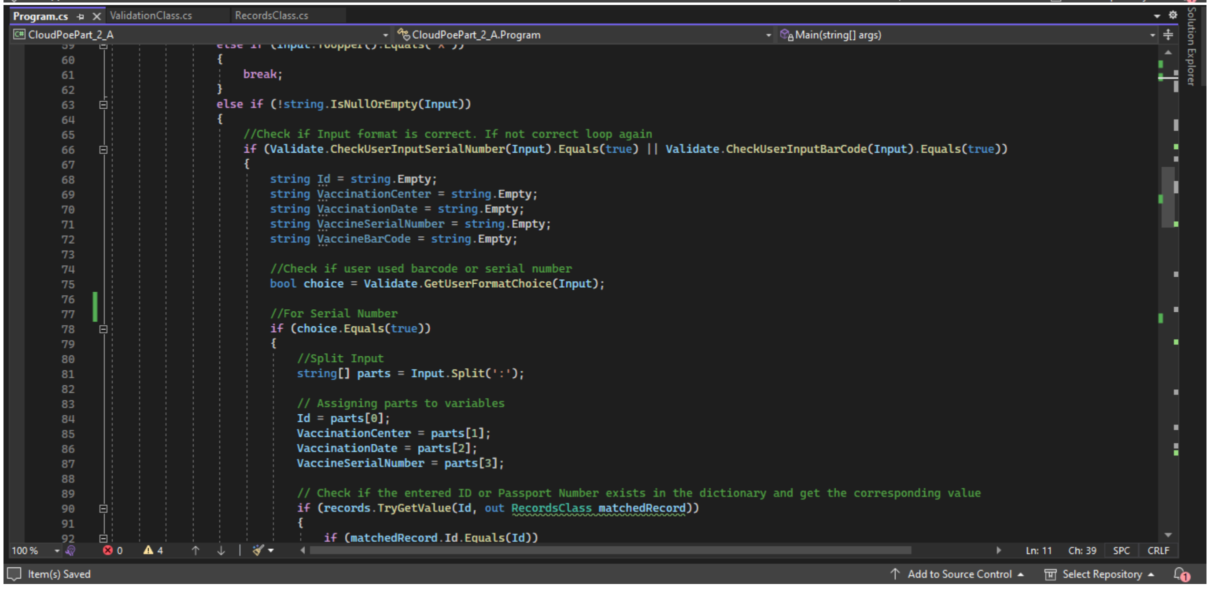
```
using System.Globalization;

namespace CloudPoePart_2_A
{
    public class ValidationClass
    {
        ///--------------------------------------------------------------///
        /// <summary>
        /// Default Constructor
        /// </summary>
        public ValidationClass()
        {

        }

        ///--------------------------------------------------------------///
        /// <summary>
        /// Method to check for valid SA passport number
        /// Checks if string is 9 characters long
        /// Checks if first character is 'A', 'E' or 'M'
        /// Checks if second character is 0
        /// Checks if the next 8 characters are all digits
        /// If conditions are not met it returns false
        /// </summary>
        /// <param name="passport"></param>
        /// <returns></returns>
        public bool CheckValidPassport(string passport)
        {
            if (passport.Length != 9)
            {
                return false;
            }

            if (!((passport[0] == 'A' && passport[1] == '0') ||
                (passport[0] == 'E' && passport[1] == '0') ||
                (passport[0] == 'M' && passport[1] == '0')))
            {
                return false;
            }

            for (int i = 1; i < passport.Length; i++)
            {
                if (!char.IsDigit(passport[i]))
                {
                    return false;
                }
            }

            return true;
        }
        ///-----------------------------------------------------------------///
        /// <summary>
        /// Method to check for valid Id number
        /// Check if string is 13 characters long
        /// Check if the 13 characters are all digits
        /// </summary>
        /// <param name="id"></param>
        /// <returns></returns>
        public bool CheckValidID(string id)
        {
            if (id.Length != 13)
            {
                return false;
            }

            for (int i = 0; i < id.Length; i++)
            {
                if (!char.IsDigit(id[i]))
                {
                    return false;
                }
            }

            return true;
        }

        ///-----------------------------------------------------------------///
        /// <summary>
        /// Method to check if Vaccination date format is coorect
        /// </summary>
        /// <param name="date"></param>
        /// <returns></returns>
        public bool CheckVaccinationDate(string date)
        {
            DateTime tempDate;
            return DateTime.TryParseExact(date, "dd/MM/yyyy",
CultureInfo.InvariantCulture, DateTimeStyles.None, out tempDate);
        }

        ///-----------------------------------------------------------------///
        /// <summary>
        /// Method to check if serial number is digit and 10 characters long
        /// </summary>
        /// <param name="serialNumber"></param>
        /// <returns></returns>
        public bool CheckVaccineSerialNumber(string serialNumber)
        {
            if (serialNumber.Length != 10)
            {
                return false;
            }

            return serialNumber.All(char.IsDigit);
        }

        ///-----------------------------------------------------------------///
        /// <summary>
        /// Method to check user Input is in correct format for Serial Number
        /// and validate data
        /// </summary>
        /// <param name="input"></param>
        /// <returns></returns>
        public bool CheckUserInputSerialNumber(string input)
        {
            bool Valid = false;

            string[] parts = input.Split(':');

            if (parts.Length == 4)
            {
                string VaccinationCenter = parts[1];
                string VaccinationDate = parts[2];
                string VaccineSerialNumber = parts[3];

                //Check valid date format and vaccine serial number
                if(CheckVaccinationDate(VaccinationDate).Equals(true) &&
CheckVaccineSerialNumber(VaccineSerialNumber).Equals(true))
                {
                    //Check if Vaccination center is not null
                    if (!string.IsNullOrEmpty(VaccinationCenter))
                    {
                        Valid = true;
                    }
                }
                else
                {
                    Valid = false;
                }
            }
            else
            {
                Valid = false;
            }

            return Valid;
        }

        ///----------------------------------------------------------------///
        /// <summary>
        /// Method to check user Input is in correct format for Barcode
        /// and validate data
        /// </summary>
        /// <param name="input"></param>
        /// <returns></returns>
        public bool CheckUserInputBarCode(string input)
        {
            bool Valid = false;

            string[] parts = input.Split(':');

            if (parts.Length == 4)
            {
                string VaccineBarCode = parts[0];
                string VaccinationDate = parts[1];
                string VaccinationCenter = parts[2];

                //Check valid date format and vaccine serial number
                if (CheckVaccinationDate(VaccinationDate).Equals(true) &&
CheckVaccineBarCode(VaccineBarCode).Equals(true))
                {
                    //Check if Vaccination center is not null
                    if (!string.IsNullOrEmpty(VaccinationCenter))
                    {
                        Valid = true;
                    }
                }
                else
                {
                    Valid = false;
                }
            }
            else
            {
                Valid = false;
            }

            return Valid;
        }

        ///-----------------------------------------------------------------///
        /// <summary>
        /// Method to check if barcode is digit and 12 characters long
        /// </summary>
        /// <param name="serialNumber"></param>
        /// <returns></returns>
        public bool CheckVaccineBarCode(string barCode)
        {
            if (barCode.Length != 12)
            {
                return false;
            }

            return barCode.All(char.IsDigit);
        }

        ///-----------------------------------------------------------------///
        /// <summary>
        /// Method to return user choice
        /// either Barcode or Serial number
        /// </summary>
        /// <returns></returns>
        public bool GetUserFormatChoice(string Input)
        {
            if (CheckUserInputSerialNumber(Input).Equals(true))
            {
                return true;
            }
            else
            {
                return false;
            }
        }
    }
}
//--------------------------------------------------< END >----------------------
--------------------------//

namespace CloudPoePart_2_A
{
    public class RecordsClass
    {
        /// <summary>
        /// Store ID or Passport Number
        /// </summary>
        public string Id { get; set; }

        /// <summary>
        /// Store First Name
        /// </summary>
        public string FirstName { get; set; }

        /// <summary>
        /// Store Last Name
        /// </summary>
        public string LastName { get; set; }

        /// <summary>
        /// Store Date of Birth
        /// </summary>
        public string DateOfBirth { get; set; }

        /// <summary>
        /// Store Gender
        /// </summary>
        public string Gender { get; set; }

        ///----------------------------------------------------------------///
        /// <summary>
        /// Default Constructor
        /// </summary>
        public RecordsClass()
        {

        }

        ///----------------------------------------------------------------///
        /// <summary>
        /// Parametized Constructor
        /// </summary>
        /// <param name="id"></param>
        /// <param name="vacineNumber"></param>
        /// <param name="firstName"></param>
        /// <param name="lastName"></param>
        /// <param name="dateOfBirth"></param>
        /// <param name="gender"></param>
        public RecordsClass(string id, string firstName, string lastName, string
dateOfBirth, string gender)
        {
            Id = id;
            FirstName = firstName;
            LastName = lastName;
            DateOfBirth = dateOfBirth;
            Gender = gender;
        }
    }
}
//--------------------------------------------------< END >----------------------
--------------------------//
```



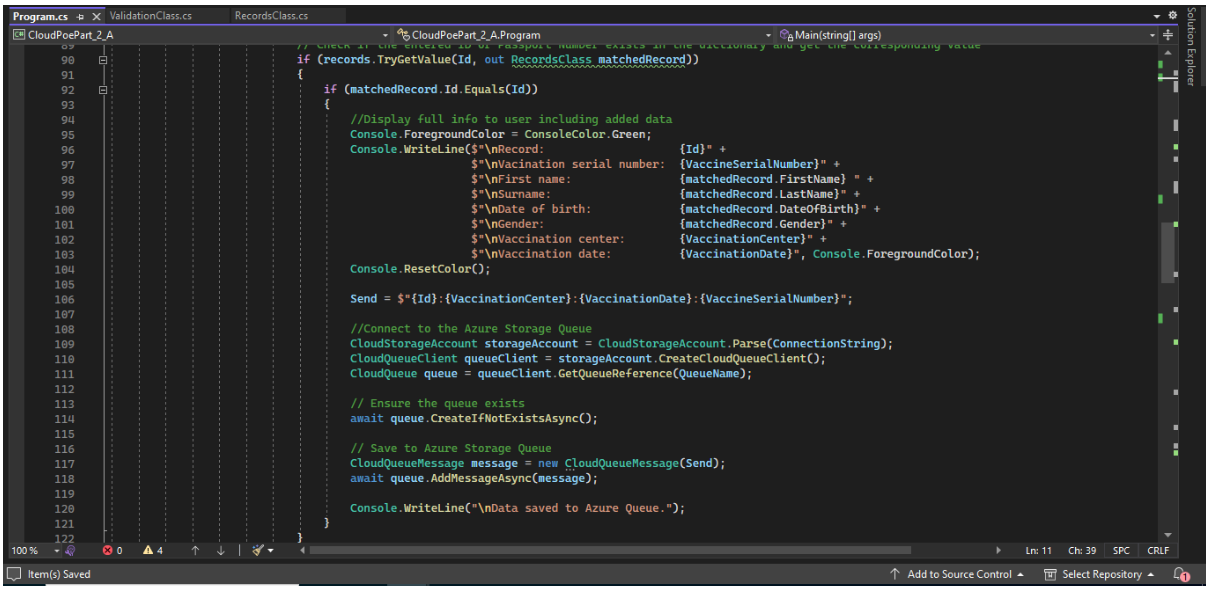
```
using Microsoft.WindowsAzure.Storage;
using Microsoft.WindowsAzure.Storage.Queue;

namespace CloudPoePart_2_A
{
    0 references
    internal class Program
    {
        0 references
        static async Task Main(string[] args)
        {
            string ConnectionString = "DefaultEndpointsProtocol=https;AccountName=queuetriggerst10025374;AccountKey=/klJ15C9P/Uq47Rg3EFhzE4kc1QkRxX" +
                                      "OVraYPPZECAoVK2TB/K8tOwbwELJ5dBC0V6I1dttdE1LX+ASteQ23hA==;EndpointSuffix=core.windows.net";
            string QueueName = "vaccinemessage";

            ValidationClass Validate = new ValidationClass();

            //Records of Passports and IDs
            Dictionary<string, RecordsClass> records = new Dictionary<string, RecordsClass>
            {
                { "A04108234", new RecordsClass( "A04108234", "Emily", "Thompson", "02/12/1986", "Female")},
                { "E04439245", new RecordsClass( "E04439245", "Jonathan", "White", "22/07/1990", "Male")},
                { "M04389256", new RecordsClass("M04389256", "Benjamin", "Carter", "03/03/1982", "Male") },
                { "9306125183012", new RecordsClass("9306125183012", "Nicole", "Johnson", "29/09/1995", "Female") },
                { "8407216123019", new RecordsClass("8407216123019", "Victoria", "Perez", "18/11/1989", "Female") },
                { "A04098378", new RecordsClass("A04098378", "Christopher", "Garcia", "15/05/1987", "Male") },
                { "A04918237", new RecordsClass("A04918237", "Samantha", "Davis", "21/04/1994", "Female") },
                { "A04829248", new RecordsClass("A04829248", "Brandon", "Nelson", "04/10/1992", "Male") },
                { "A04789259", new RecordsClass("A04789259", "Grace", "Hall", "07/08/1991", "Female") },
                { "7001012043017", new RecordsClass("7001012043017", "Joshua", "Baker", "02/12/1986", "Male") },
                { "8505053073011", new RecordsClass("8505053073011", "Rachel", "Martin", "26/06/1990", "Female") },
                { "7609107583014", new RecordsClass("7609107583014", "Hannah", "Young", "13/03/1989", "Female") },
            };
```



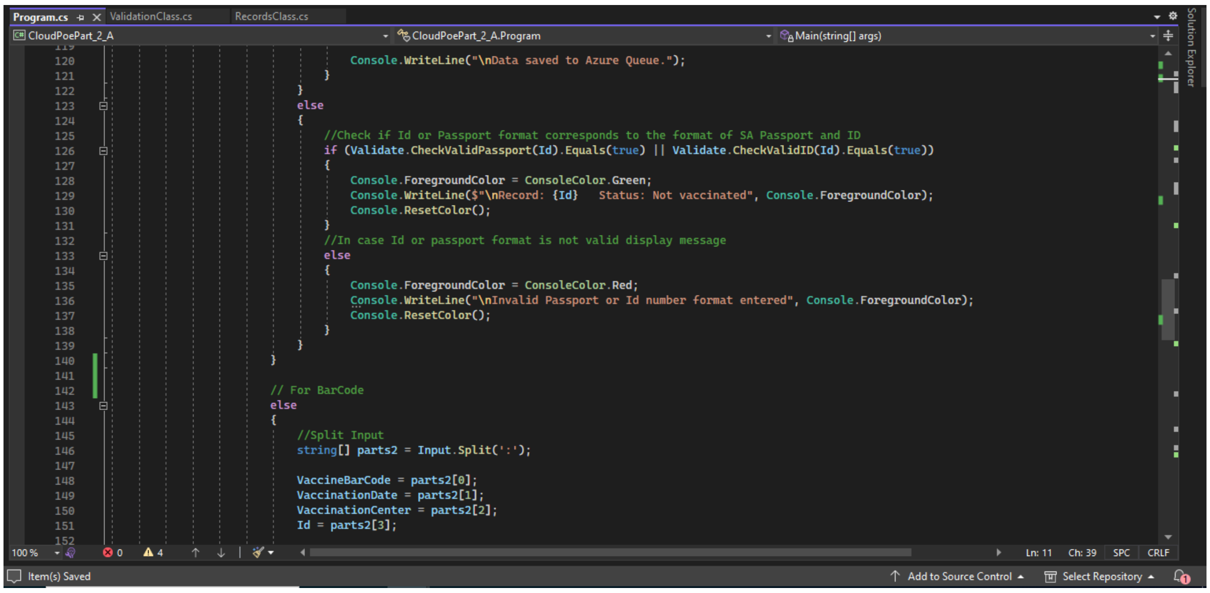
```
                { "7609107583014", new RecordsClass("7609107583014", "Hannah", "Young", "13/03/1989", "Female") },
            };

            Console.WriteLine("----- VACCINE VALIDATION SYSTEM -----");

            string Input = string.Empty;

            //Store info to be sent to Queue
            string Send = string.Empty;

            do
            {
                //Ask user Input
                Console.WriteLine("\nProvide the data for the id or passport record (Type X to exit)" +
                                  "\nNote: Type the data in this formats (Id:VaccinationCenter:VaccinationDate[dd/mm/yyyy]:VaccineSerialNumber)" +
                                  "\n                                  (VaccineBarcode:VaccinationDate[dd/mm/yyyy]:VaccinationCenter:Id)\n");

                //Get Input
                Input = Console.ReadLine();

                //Check for Null and Empty
                //Return message in case validation fails
                if (string.IsNullOrEmpty(Input))
                {
                    Console.ForegroundColor = ConsoleColor.Red;
                    Console.WriteLine("\nPlease provide the data", Console.ForegroundColor);
                    Console.ResetColor();
                }
                //In case user enters X exit the program
                else if (Input.ToUpper().Equals("X"))
                {
                    break;
                }
```



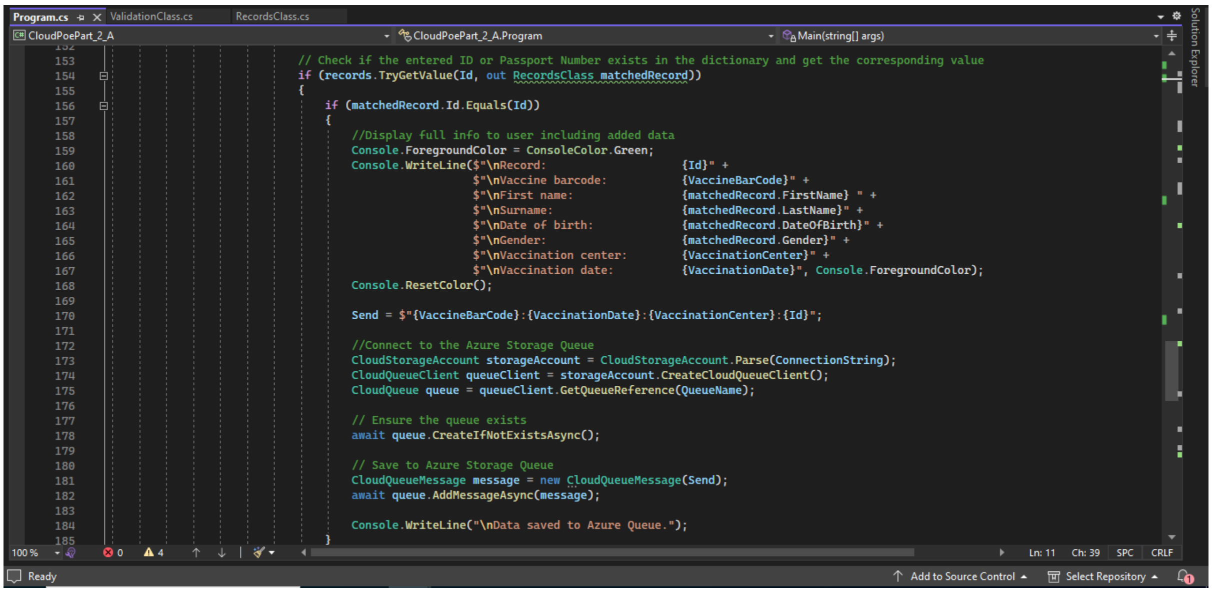
```
                {
                    break;
                }
                else if (!string.IsNullOrEmpty(Input))
                {
                    //Check if Input format is correct. If not correct loop again
                    if (Validate.CheckUserInputSerialNumber(Input).Equals(true) || Validate.CheckUserInputBarCode(Input).Equals(true))
                    {
                        string Id = string.Empty;
                        string VaccinationCenter = string.Empty;
                        string VaccinationDate = string.Empty;
                        string VaccineSerialNumber = string.Empty;
                        string VaccineBarCode = string.Empty;

                        //Check if user used barcode or serial number
                        bool choice = Validate.GetUserFormatChoice(Input);

                        //For Serial Number
                        if (choice.Equals(true))
                        {
                            //Split Input
                            string[] parts = Input.Split(':');

                            // Assigning parts to variables
                            Id = parts[0];
                            VaccinationCenter = parts[1];
                            VaccinationDate = parts[2];
                            VaccineSerialNumber = parts[3];

                            // Check if the entered ID or Passport Number exists in the dictionary and get the corresponding value
                            if (records.TryGetValue(Id, out RecordsClass matchedRecord))
                            {
                                if (matchedRecord.Id.Equals(Id))
```



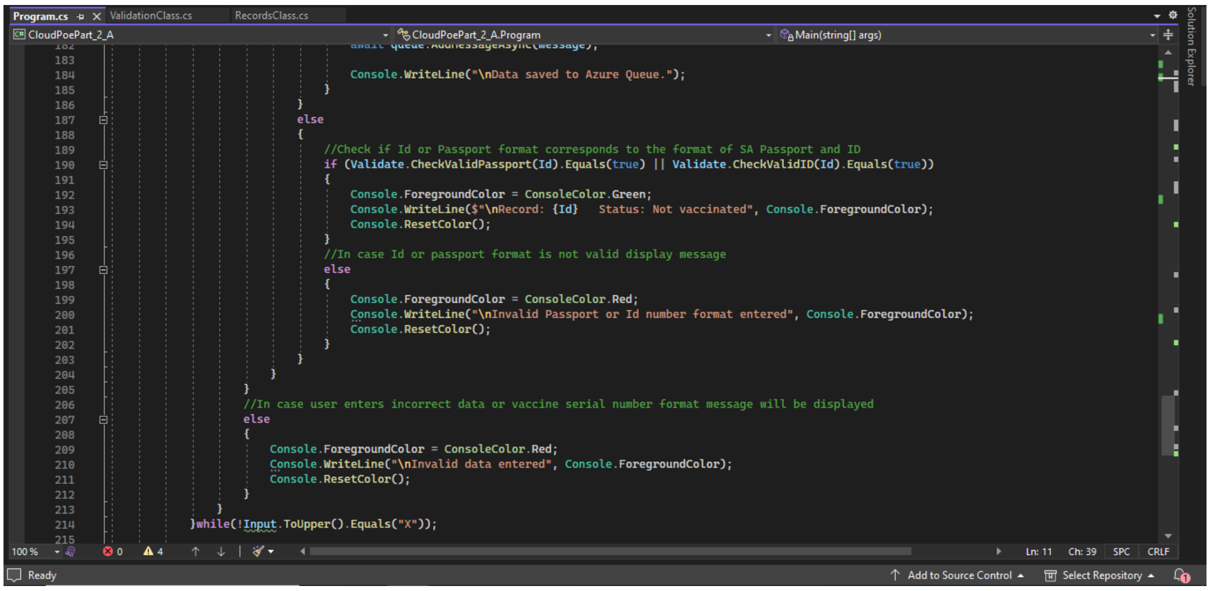
```
    // Check if the entered ID or Passport Number exists in the dictionary and get the corresponding value
    if (records.TryGetValue(Id, out RecordsClass matchedRecord))
    {
        if (matchedRecord.Id.Equals(Id))
        {
            //Display full info to user including added data
            Console.ForegroundColor = ConsoleColor.Green;
            Console.WriteLine($"\nRecord:                       {Id}" +
                              $"\nVacination serial number:  {VaccineSerialNumber}" +
                              $"\nFirst name:                {matchedRecord.FirstName} " +
                              $"\nSurname:                   {matchedRecord.LastName}" +
                              $"\nDate of birth:             {matchedRecord.DateOfBirth}" +
                              $"\nGender:                    {matchedRecord.Gender}" +
                              $"\nVaccination center:        {VaccinationCenter}" +
                              $"\nVaccination date:          {VaccinationDate}", Console.ForegroundColor);
            Console.ResetColor();

            Send = $"{Id}:{VaccinationCenter}:{VaccinationDate}:{VaccineSerialNumber}";

            //Connect to the Azure Storage Queue
            CloudStorageAccount storageAccount = CloudStorageAccount.Parse(ConnectionString);
            CloudQueueClient queueClient = storageAccount.CreateCloudQueueClient();
            CloudQueue queue = queueClient.GetQueueReference(QueueName);

            // Ensure the queue exists
            await queue.CreateIfNotExistsAsync();

            // Save to Azure Storage Queue
            CloudQueueMessage message = new CloudQueueMessage(Send);
            await queue.AddMessageAsync(message);

            Console.WriteLine("\nData saved to Azure Queue.");
        }
    }
```



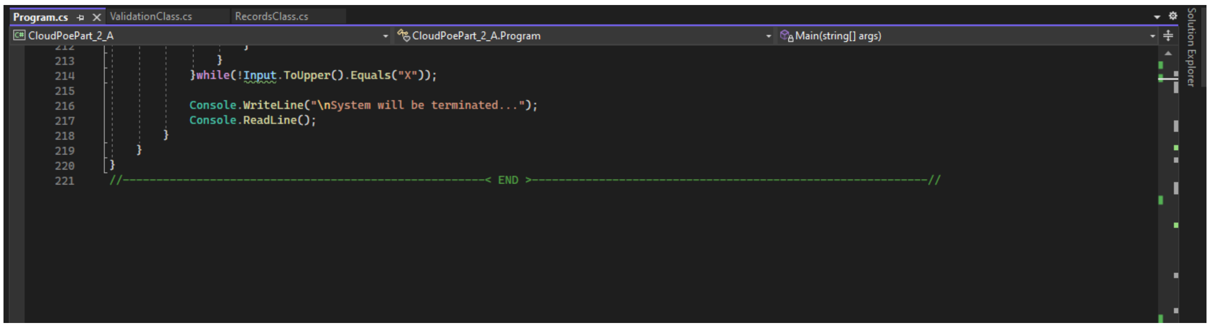
```
            Console.WriteLine("\nData saved to Azure Queue.");
        }
    }
    else
    {
        //Check if Id or Passport format corresponds to the format of SA Passport and ID
        if (Validate.CheckValidPassport(Id).Equals(true) || Validate.CheckValidID(Id).Equals(true))
        {
            Console.ForegroundColor = ConsoleColor.Green;
            Console.WriteLine($"\nRecord: {Id}   Status: Not vaccinated", Console.ForegroundColor);
            Console.ResetColor();
        }
        //In case Id or passport format is not valid display message
        else
        {
            Console.ForegroundColor = ConsoleColor.Red;
            Console.WriteLine("\nInvalid Passport or Id number format entered", Console.ForegroundColor);
            Console.ResetColor();
        }
    }
}

// For BarCode
else
{
    //Split Input
    string[] parts2 = Input.Split(':');

    VaccineBarCode = parts2[0];
    VaccinationDate = parts2[1];
    VaccinationCenter = parts2[2];
    Id = parts2[3];
```



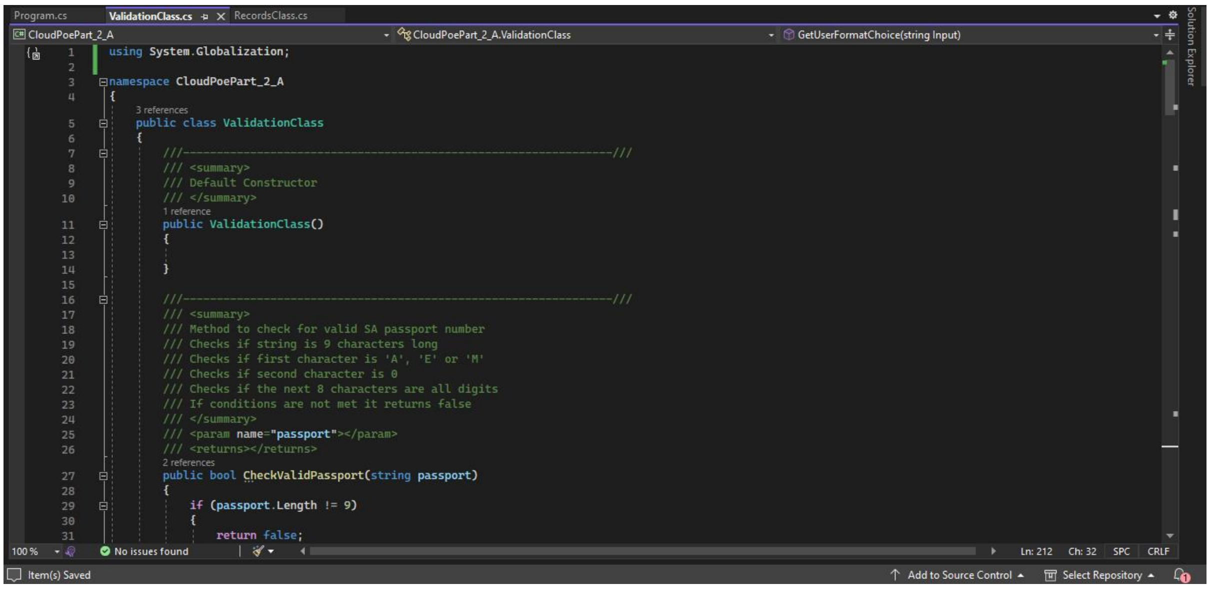
```
    // Check if the entered ID or Passport Number exists in the dictionary and get the corresponding value
    if (records.TryGetValue(Id, out RecordsClass matchedRecord))
    {
        if (matchedRecord.Id.Equals(Id))
        {
            //Display full info to user including added data
            Console.ForegroundColor = ConsoleColor.Green;
            Console.WriteLine($"\nRecord:                       {Id}" +
                              $"\nVaccine barcode:           {VaccineBarCode}" +
                              $"\nFirst name:                {matchedRecord.FirstName} " +
                              $"\nSurname:                   {matchedRecord.LastName}" +
                              $"\nDate of birth:             {matchedRecord.DateOfBirth}" +
                              $"\nGender:                    {matchedRecord.Gender}" +
                              $"\nVaccination center:        {VaccinationCenter}" +
                              $"\nVaccination date:          {VaccinationDate}", Console.ForegroundColor);
            Console.ResetColor();

            Send = $"{VaccineBarCode}:{VaccinationDate}:{VaccinationCenter}:{Id}";

            //Connect to the Azure Storage Queue
            CloudStorageAccount storageAccount = CloudStorageAccount.Parse(ConnectionString);
            CloudQueueClient queueClient = storageAccount.CreateCloudQueueClient();
            CloudQueue queue = queueClient.GetQueueReference(QueueName);

            // Ensure the queue exists
            await queue.CreateIfNotExistsAsync();

            // Save to Azure Storage Queue
            CloudQueueMessage message = new CloudQueueMessage(Send);
            await queue.AddMessageAsync(message);

            Console.WriteLine("\nData saved to Azure Queue.");
        }
```



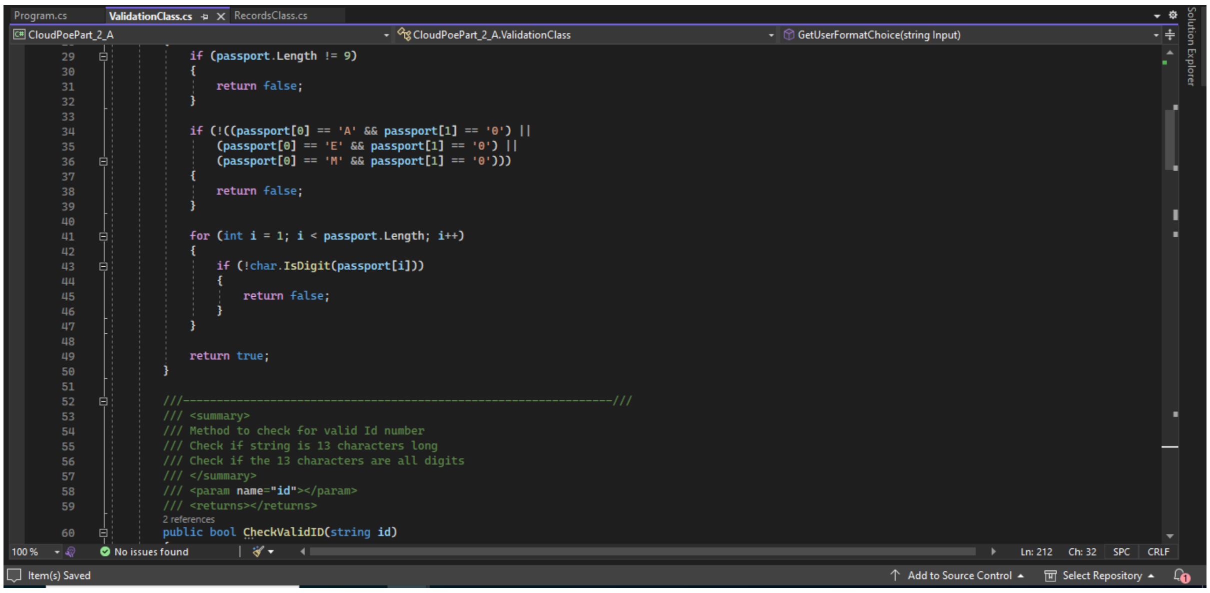
```
                                    await queue.AddMessageAsync(message);

                                    Console.WriteLine("\nData saved to Azure Queue.");
                                }
                            }
                            else
                            {
                                //Check if Id or Passport format corresponds to the format of SA Passport and ID
                                if (Validate.CheckValidPassport(Id).Equals(true) || Validate.CheckValidID(Id).Equals(true))
                                {
                                    Console.ForegroundColor = ConsoleColor.Green;
                                    Console.WriteLine($"\nRecord: {Id}   Status: Not vaccinated", Console.ForegroundColor);
                                    Console.ResetColor();
                                }
                                //In case Id or passport format is not valid display message
                                else
                                {
                                    Console.ForegroundColor = ConsoleColor.Red;
                                    Console.WriteLine("\nInvalid Passport or Id number format entered", Console.ForegroundColor);
                                    Console.ResetColor();
                                }
                            }
                        }
                    }
                    //In case user enters incorrect data or vaccine serial number format message will be displayed
                    else
                    {
                        Console.ForegroundColor = ConsoleColor.Red;
                        Console.WriteLine("\nInvalid data entered", Console.ForegroundColor);
                        Console.ResetColor();
                    }
                }
            }while(!Input.ToUpper().Equals("X"));

```



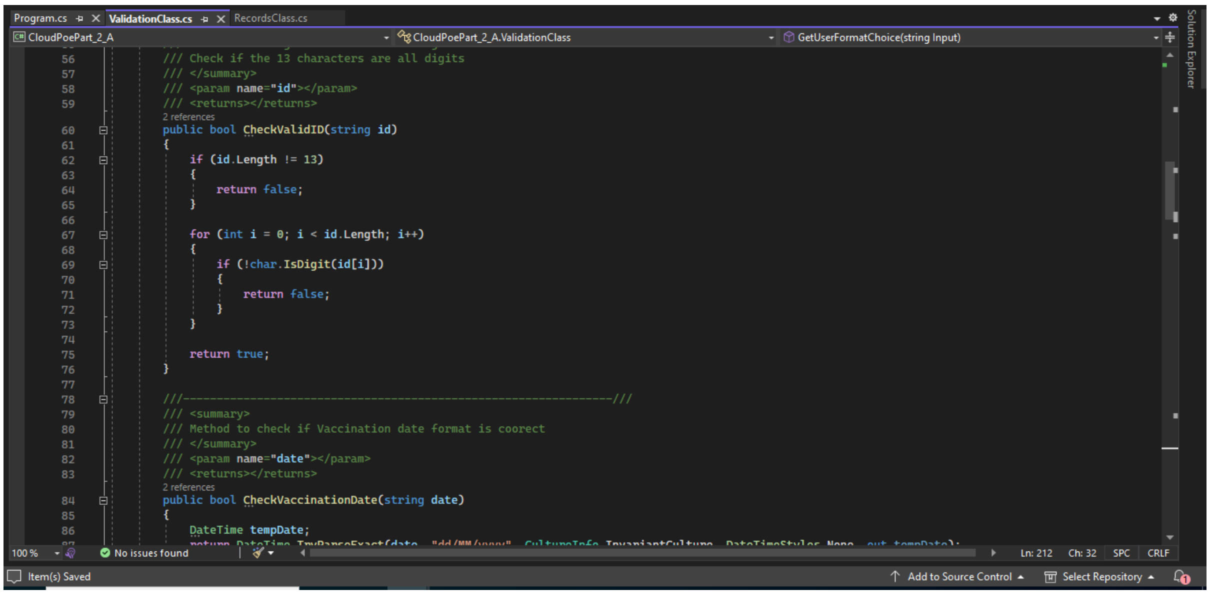
```
                }
            }while(!Input.ToUpper().Equals("X"));

            Console.WriteLine("\nSystem will be terminated...");
            Console.ReadLine();
        }
    }
}
//------------------------------------------------------< END >----------------------------------------------------------//
```



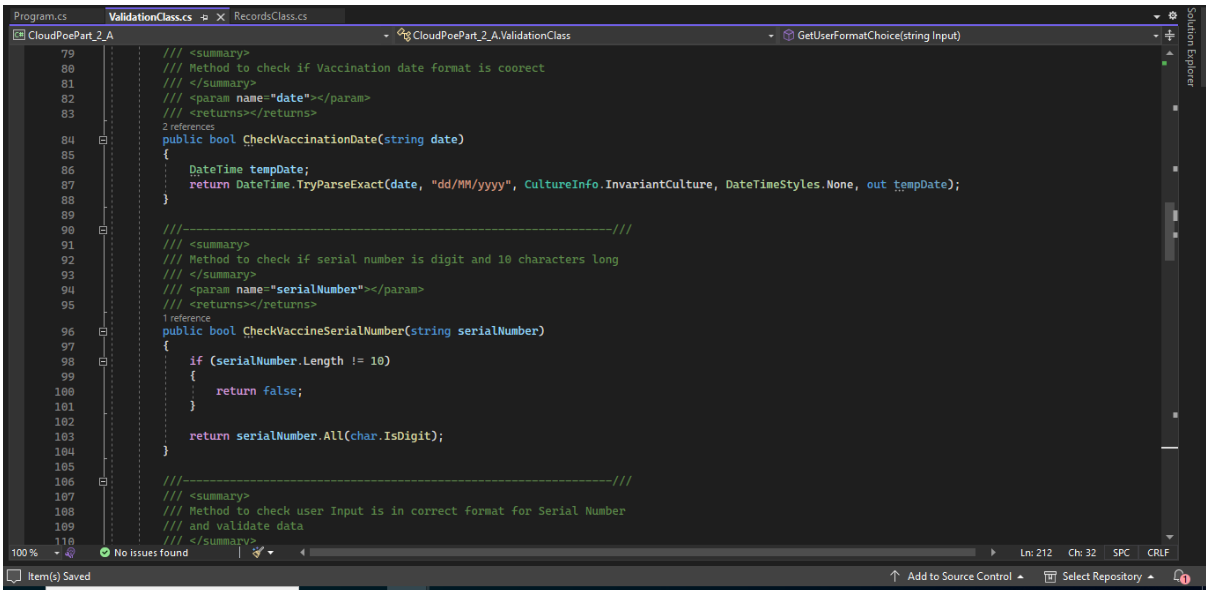
```
using System.Globalization;

namespace CloudPoePart_2_A
{
    3 references
    public class ValidationClass
    {
        ///----------------------------------------------------------------///
        /// <summary>
        /// Default Constructor
        /// </summary>
        1 reference
        public ValidationClass()
        {

        }

        ///----------------------------------------------------------------///
        /// <summary>
        /// Method to check for valid SA passport number
        /// Checks if string is 9 characters long
        /// Checks if first character is 'A', 'E' or 'M'
        /// Checks if second character is 0
        /// Checks if the next 8 characters are all digits
        /// If conditions are not met it returns false
        /// </summary>
        /// <param name="passport"></param>
        /// <returns></returns>
        2 references
        public bool CheckValidPassport(string passport)
        {
            if (passport.Length != 9)
            {
                return false;
```



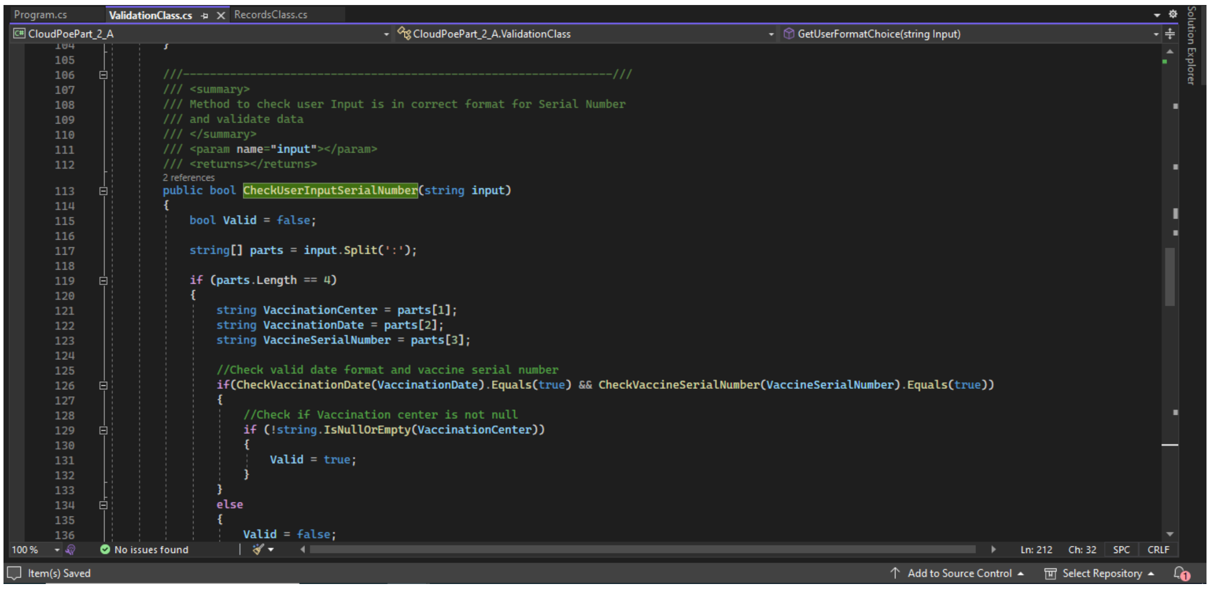
```
            if (passport.Length != 9)
            {
                return false;
            }

            if (!((passport[0] == 'A' && passport[1] == '0') ||
                (passport[0] == 'E' && passport[1] == '0') ||
                (passport[0] == 'M' && passport[1] == '0')))
            {
                return false;
            }

            for (int i = 1; i < passport.Length; i++)
            {
                if (!char.IsDigit(passport[i]))
                {
                    return false;
                }
            }

            return true;
        }

        ///---------------------------------------------------------------///
        /// <summary>
        /// Method to check for valid Id number
        /// Check if string is 13 characters long
        /// Check if the 13 characters are all digits
        /// </summary>
        /// <param name="id"></param>
        /// <returns></returns>
        2 references
        public bool CheckValidID(string id)
```



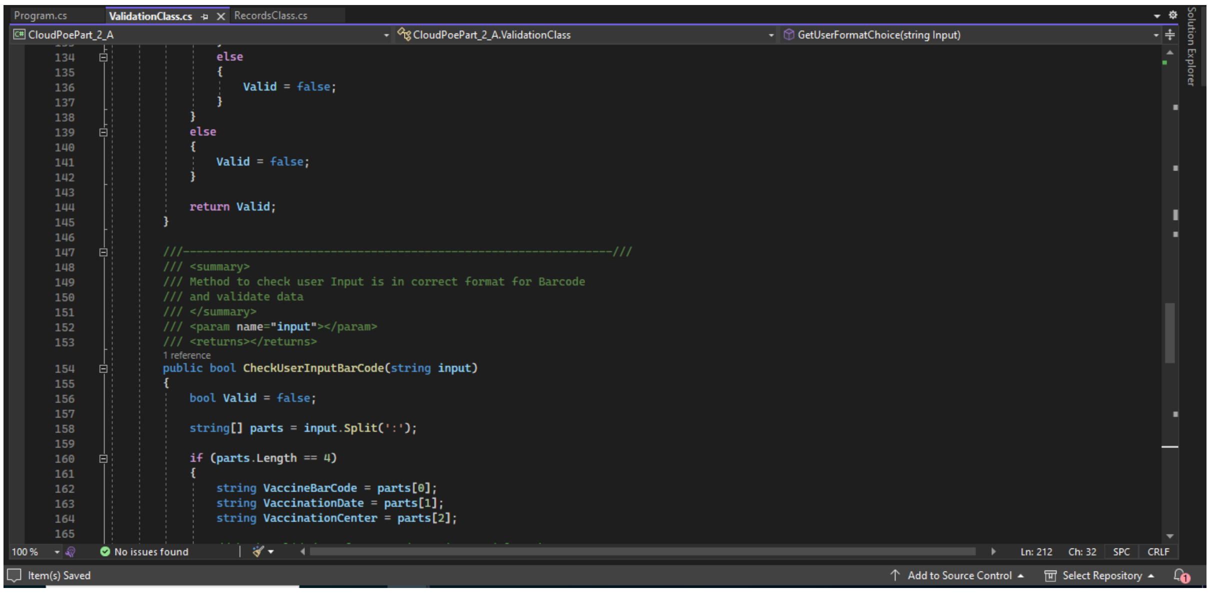
```
        /// Check if the 13 characters are all digits
        /// </summary>
        /// <param name="id"></param>
        /// <returns></returns>
        2 references
        public bool CheckValidID(string id)
        {
            if (id.Length != 13)
            {
                return false;
            }

            for (int i = 0; i < id.Length; i++)
            {
                if (!char.IsDigit(id[i]))
                {
                    return false;
                }
            }

            return true;
        }

        ///---------------------------------------------------------------///
        /// <summary>
        /// Method to check if Vaccination date format is coorect
        /// </summary>
        /// <param name="date"></param>
        /// <returns></returns>
        2 references
        public bool CheckVaccinationDate(string date)
        {
            DateTime tempDate;
            return DateTime.TryParseExact(date, "dd/MM/yyyy", CultureInfo.InvariantCulture, DateTimeStyles.None, out tempDate);
```



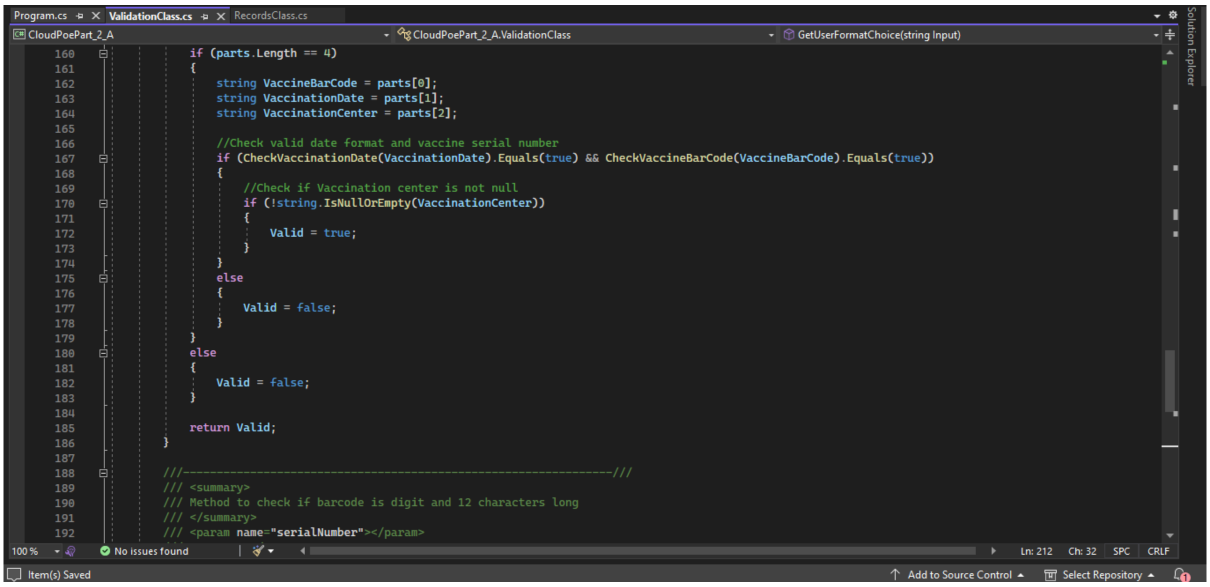
```
        /// <summary>
        /// Method to check if Vaccination date format is coorect
        /// </summary>
        /// <param name="date"></param>
        /// <returns></returns>
        2 references
        public bool CheckVaccinationDate(string date)
        {
            DateTime tempDate;
            return DateTime.TryParseExact(date, "dd/MM/yyyy", CultureInfo.InvariantCulture, DateTimeStyles.None, out tempDate);
        }

        ///---------------------------------------------------------------///
        /// <summary>
        /// Method to check if serial number is digit and 10 characters long
        /// </summary>
        /// <param name="serialNumber"></param>
        /// <returns></returns>
        1 reference
        public bool CheckVaccineSerialNumber(string serialNumber)
        {
            if (serialNumber.Length != 10)
            {
                return false;
            }

            return serialNumber.All(char.IsDigit);
        }

        ///---------------------------------------------------------------///
        /// <summary>
        /// Method to check user Input is in correct format for Serial Number
        /// and validate data
        /// </summary>
```



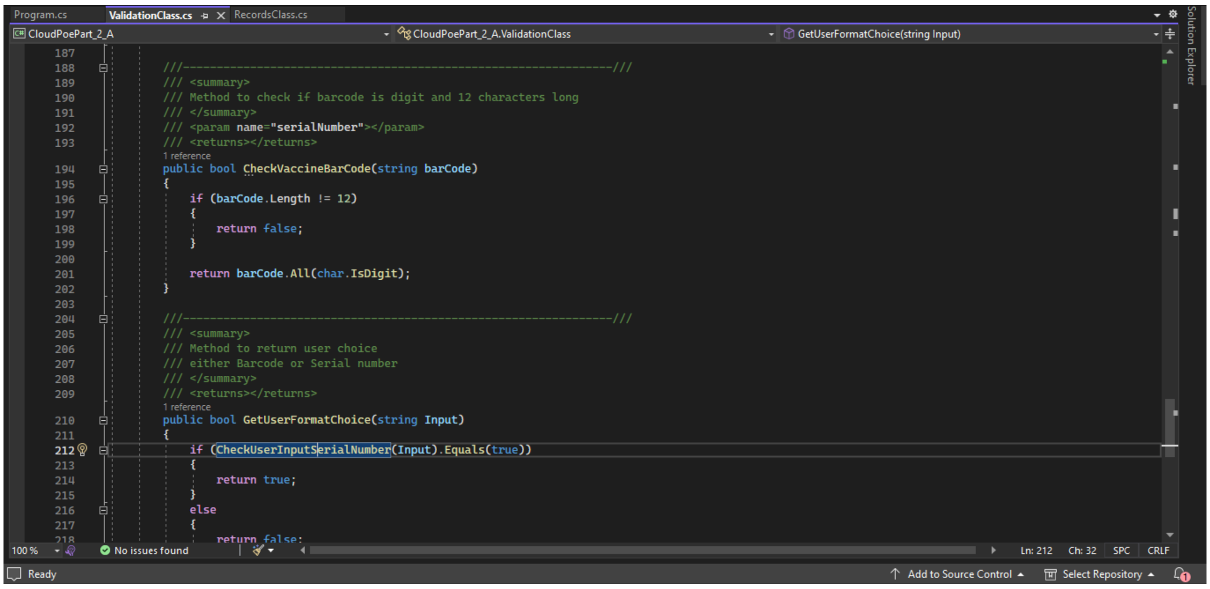
```
    }

    ///---------------------------------------------------------------///
    /// <summary>
    /// Method to check user Input is in correct format for Serial Number
    /// and validate data
    /// </summary>
    /// <param name="input"></param>
    /// <returns></returns>
    2 references
    public bool CheckUserInputSerialNumber(string input)
    {
        bool Valid = false;

        string[] parts = input.Split(':');

        if (parts.Length == 4)
        {
            string VaccinationCenter = parts[1];
            string VaccinationDate = parts[2];
            string VaccineSerialNumber = parts[3];

            //Check valid date format and vaccine serial number
            if(CheckVaccinationDate(VaccinationDate).Equals(true) && CheckVaccineSerialNumber(VaccineSerialNumber).Equals(true))
            {
                //Check if Vaccination center is not null
                if (!string.IsNullOrEmpty(VaccinationCenter))
                {
                    Valid = true;
                }
            }
            else
            {
                Valid = false;
```



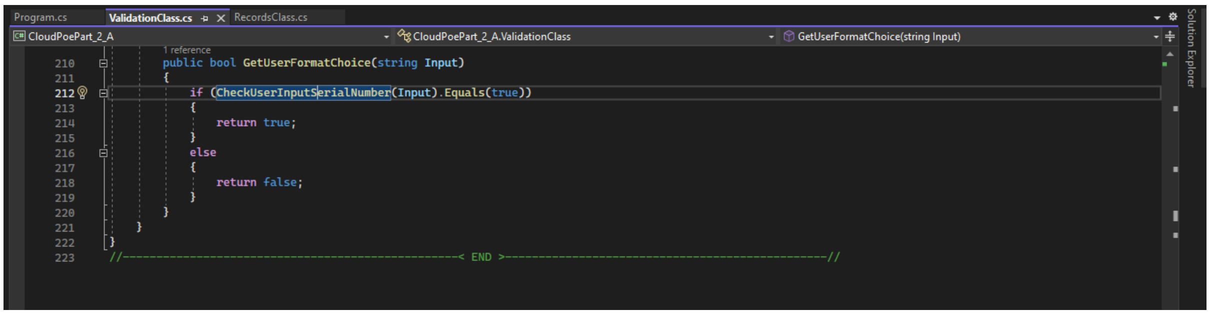
```
            else
            {
                Valid = false;
            }
        }
        else
        {
            Valid = false;
        }

        return Valid;
    }

    ///---------------------------------------------------------------///
    /// <summary>
    /// Method to check user Input is in correct format for Barcode
    /// and validate data
    /// </summary>
    /// <param name="input"></param>
    /// <returns></returns>
    1 reference
    public bool CheckUserInputBarCode(string input)
    {
        bool Valid = false;

        string[] parts = input.Split(':');

        if (parts.Length == 4)
        {
            string VaccineBarCode = parts[0];
            string VaccinationDate = parts[1];
            string VaccinationCenter = parts[2];

```



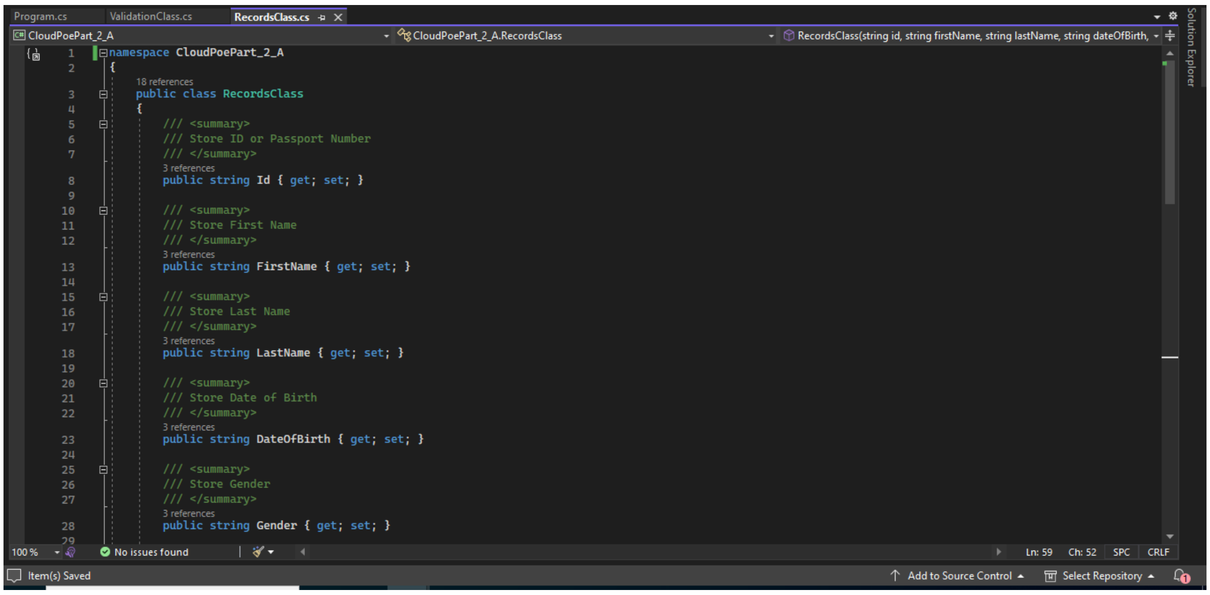
```
        if (parts.Length == 4)
        {
            string VaccineBarCode = parts[0];
            string VaccinationDate = parts[1];
            string VaccinationCenter = parts[2];

            //Check valid date format and vaccine serial number
            if (CheckVaccinationDate(VaccinationDate).Equals(true) && CheckVaccineBarCode(VaccineBarCode).Equals(true))
            {
                //Check if Vaccination center is not null
                if (!string.IsNullOrEmpty(VaccinationCenter))
                {
                    Valid = true;
                }
            }
            else
            {
                Valid = false;
            }
        }
        else
        {
            Valid = false;
        }

        return Valid;
    }

    ///---------------------------------------------------------------///
    /// <summary>
    /// Method to check if barcode is digit and 12 characters long
    /// </summary>
    /// <param name="serialNumber"></param>
```



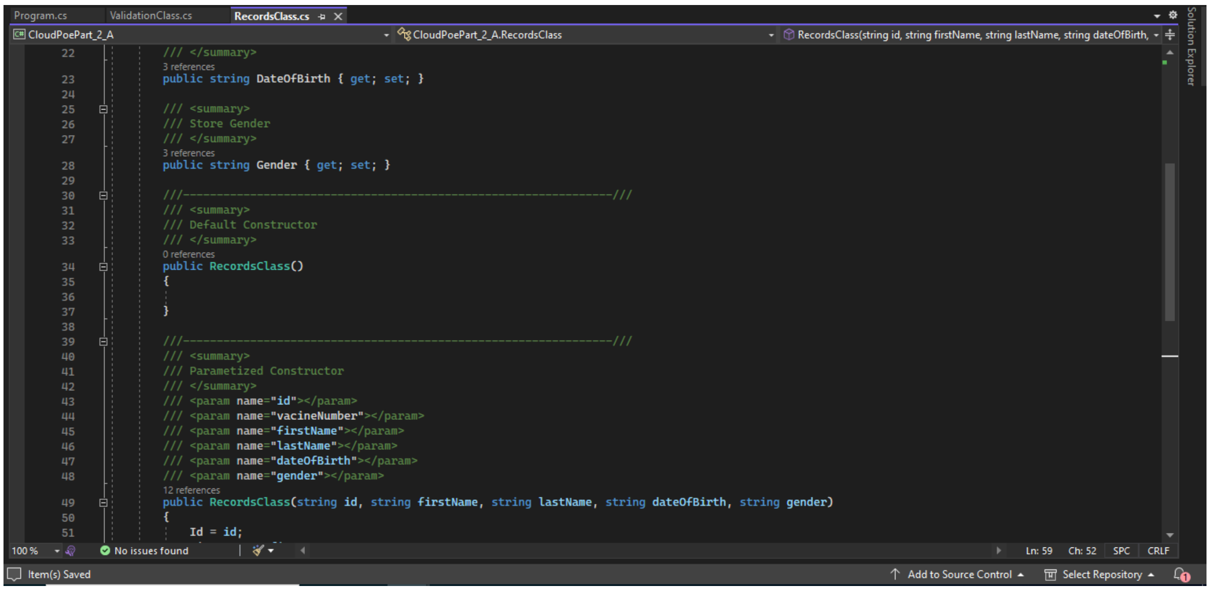
```
        ///-------------------------------------------------------------///
        /// <summary>
        /// Method to check if barcode is digit and 12 characters long
        /// </summary>
        /// <param name="serialNumber"></param>
        /// <returns></returns>
        1 reference
        public bool CheckVaccineBarCode(string barCode)
        {
            if (barCode.Length != 12)
            {
                return false;
            }

            return barCode.All(char.IsDigit);
        }

        ///-------------------------------------------------------------///
        /// <summary>
        /// Method to return user choice
        /// either Barcode or Serial number
        /// </summary>
        /// <returns></returns>
        1 reference
        public bool GetUserFormatChoice(string Input)
        {
            if (CheckUserInputSerialNumber(Input).Equals(true))
            {
                return true;
            }
            else
            {
                return false;
```



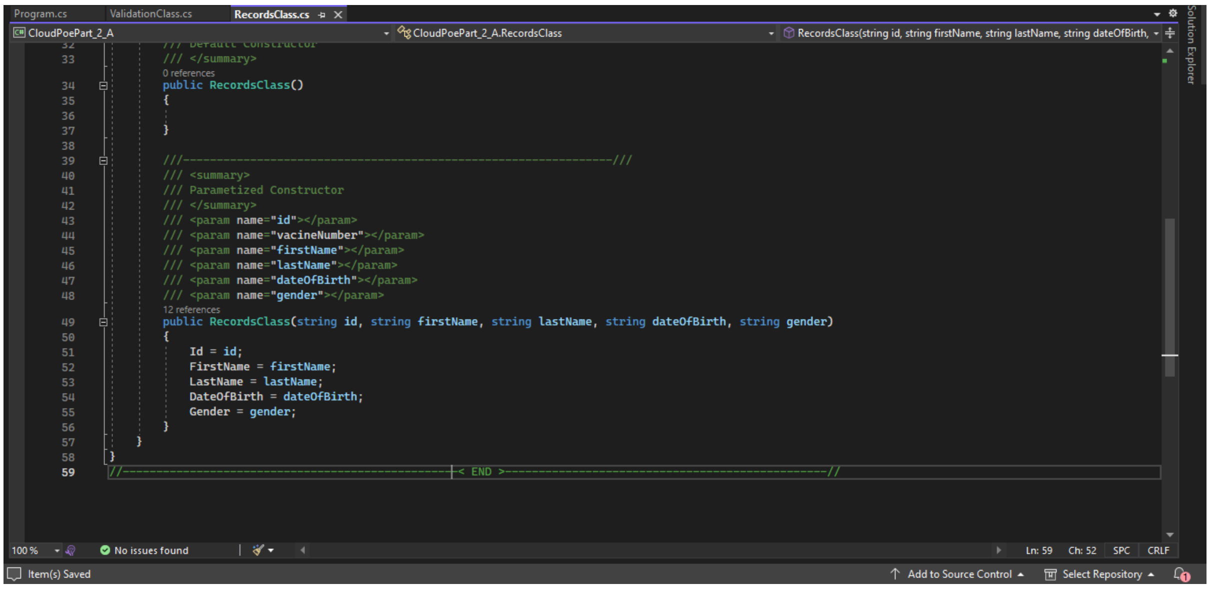
```
        1 reference
        public bool GetUserFormatChoice(string Input)
        {
            if (CheckUserInputSerialNumber(Input).Equals(true))
            {
                return true;
            }
            else
            {
                return false;
            }
        }
    }
}
//-------------------------------------------------< END >----------------------------------------------//
```



```
namespace CloudPoePart_2_A
{
    18 references
    public class RecordsClass
    {
        /// <summary>
        /// Store ID or Passport Number
        /// </summary>
        3 references
        public string Id { get; set; }

        /// <summary>
        /// Store First Name
        /// </summary>
        3 references
        public string FirstName { get; set; }

        /// <summary>
        /// Store Last Name
        /// </summary>
        3 references
        public string LastName { get; set; }

        /// <summary>
        /// Store Date of Birth
        /// </summary>
        3 references
        public string DateOfBirth { get; set; }

        /// <summary>
        /// Store Gender
        /// </summary>
        3 references
        public string Gender { get; set; }
```



```
        /// </summary>
        3 references
        public string DateOfBirth { get; set; }

        /// <summary>
        /// Store Gender
        /// </summary>
        3 references
        public string Gender { get; set; }

        ///-----------------------------------------------------------------///
        /// <summary>
        /// Default Constructor
        /// </summary>
        0 references
        public RecordsClass()
        {

        }

        ///-----------------------------------------------------------------///
        /// <summary>
        /// Parametized Constructor
        /// </summary>
        /// <param name="id"></param>
        /// <param name="vacineNumber"></param>
        /// <param name="firstName"></param>
        /// <param name="lastName"></param>
        /// <param name="dateOfBirth"></param>
        /// <param name="gender"></param>
        12 references
        public RecordsClass(string id, string firstName, string lastName, string dateOfBirth, string gender)
        {
            Id = id;
```

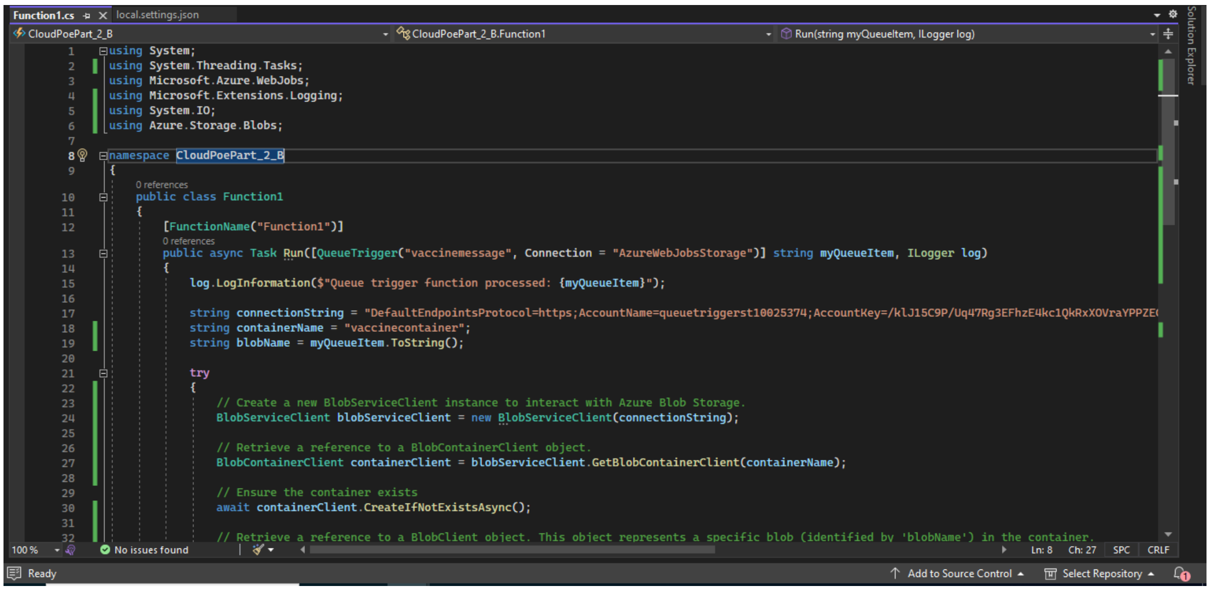


```
        /// </summary>
        0 references
        public RecordsClass()
        {

        }

        ///-----------------------------------------------------------------///
        /// <summary>
        /// Parametized Constructor
        /// </summary>
        /// <param name="id"></param>
        /// <param name="vacineNumber"></param>
        /// <param name="firstName"></param>
        /// <param name="lastName"></param>
        /// <param name="dateOfBirth"></param>
        /// <param name="gender"></param>
        12 references
        public RecordsClass(string id, string firstName, string lastName, string dateOfBirth, string gender)
        {
            Id = id;
            FirstName = firstName;
            LastName = lastName;
            DateOfBirth = dateOfBirth;
            Gender = gender;
        }
    }
}
//--------------------------------------------------< END >--------------------------------------------//
```

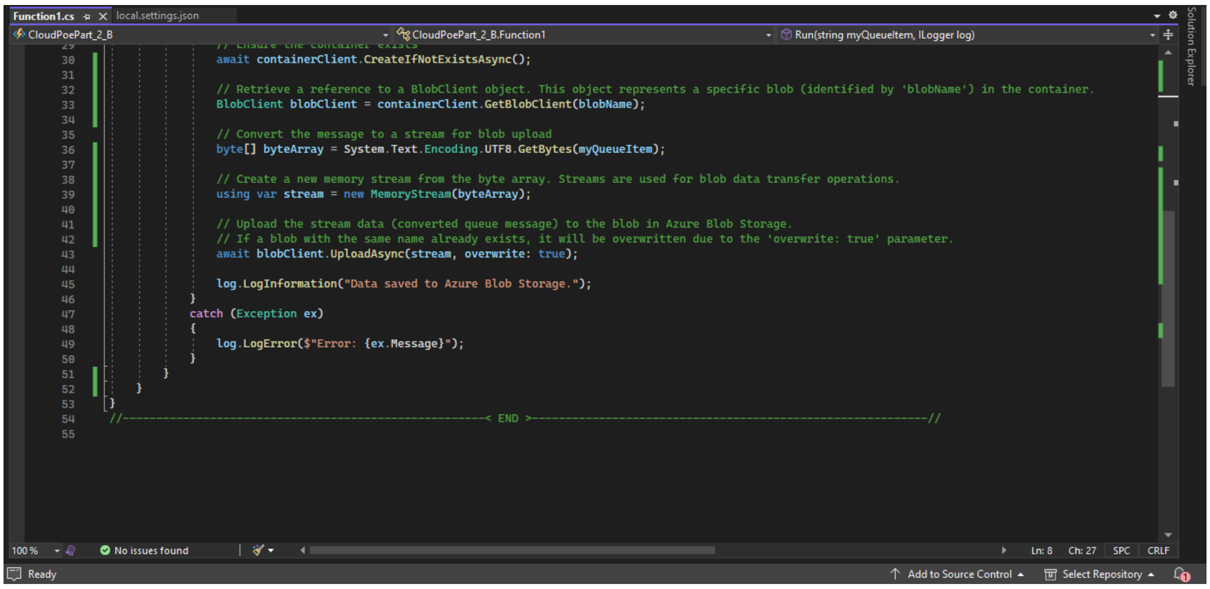
Function Code

```
using System;
using System.Threading.Tasks;
using Microsoft.Azure.WebJobs;
using Microsoft.Extensions.Logging;
using System.IO;
using Azure.Storage.Blobs;

namespace CloudPoePart_2_B
{
    public class Function1
    {
        [FunctionName("Function1")]
        public async Task Run([QueueTrigger("vaccinemessage", Connection =
"AzureWebJobsStorage")] string myQueueItem, ILogger log)
        {
            log.LogInformation($"Queue trigger function processed:
{myQueueItem}");

            string connectionString =
"DefaultEndpointsProtocol=https;AccountName=queuetriggerst10025374;AccountKey=/kl
J15C9P/Uq47Rg3EFhzE4kc1QkRxXOVraYPPZECAoVK2TB/K8tOwbwELJ5dBC0V6I1dttdE1LX+ASteQ23
hA==;EndpointSuffix=core.windows.net";
            string containerName = "vaccinecontainer";
            string blobName = myQueueItem.ToString();

            try
            {
                // Create a new BlobServiceClient instance to interact with Azure
Blob Storage.
                BlobServiceClient blobServiceClient = new
BlobServiceClient(connectionString);

                // Retrieve a reference to a BlobContainerClient object.
                BlobContainerClient containerClient =
blobServiceClient.GetBlobContainerClient(containerName);

                // Ensure the container exists
                await containerClient.CreateIfNotExistsAsync();

                // Retrieve a reference to a BlobClient object. This object
represents a specific blob (identified by 'blobName') in the container.
                BlobClient blobClient = containerClient.GetBlobClient(blobName);

                // Convert the message to a stream for blob upload
                byte[] byteArray =
System.Text.Encoding.UTF8.GetBytes(myQueueItem);

                // Create a new memory stream from the byte array. Streams are
used for blob data transfer operations.
                using var stream = new MemoryStream(byteArray);

                // Upload the stream data (converted queue message) to the blob
in Azure Blob Storage.
                // If a blob with the same name already exists, it will be
overwritten due to the 'overwrite: true' parameter.
                await blobClient.UploadAsync(stream, overwrite: true);

                log.LogInformation("Data saved to Azure Blob Storage.");
            }
            catch (Exception ex)
            {
                log.LogError($"Error: {ex.Message}");
            }
        }
    }
}
//-----------------------------------------------------< END >------------------
-----------------------------------------//
```

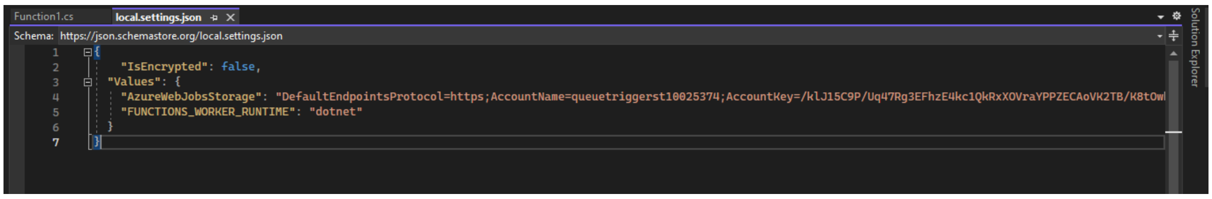
```
{
    "IsEncrypted": false,
  "Values": {
    "AzureWebJobsStorage":
"DefaultEndpointsProtocol=https;AccountName=queuetriggerst10025374;AccountKey=/kl
J15C9P/Uq47Rg3EFhzE4kc1QkRxXOVraYPPZECAoVK2TB/K8tOwbwELJ5dBC0V6I1dttdE1LX+ASteQ23
hA==;EndpointSuffix=core.windows.net",
    "FUNCTIONS_WORKER_RUNTIME": "dotnet"
  }
}
```



```csharp
using System;
using System.Threading.Tasks;
using Microsoft.Azure.WebJobs;
using Microsoft.Extensions.Logging;
using System.IO;
using Azure.Storage.Blobs;

namespace CloudPoePart_2_B
{
    0 references
    public class Function1
    {
        [FunctionName("Function1")]
        0 references
        public async Task Run([QueueTrigger("vaccinemessage", Connection = "AzureWebJobsStorage")] string myQueueItem, ILogger log)
        {
            log.LogInformation($"Queue trigger function processed: {myQueueItem}");

            string connectionString = "DefaultEndpointsProtocol=https;AccountName=queuetriggerst10025374;AccountKey=/klJ15C9P/Uq47Rg3EFhzE4kc1QkRxXOVraYPPZE
            string containerName = "vaccinecontainer";
            string blobName = myQueueItem.ToString();

            try
            {
                // Create a new BlobServiceClient instance to interact with Azure Blob Storage.
                BlobServiceClient blobServiceClient = new BlobServiceClient(connectionString);

                // Retrieve a reference to a BlobContainerClient object.
                BlobContainerClient containerClient = blobServiceClient.GetBlobContainerClient(containerName);

                // Ensure the container exists
                await containerClient.CreateIfNotExistsAsync();

                // Retrieve a reference to a BlobClient object. This object represents a specific blob (identified by 'blobName') in the container.
```

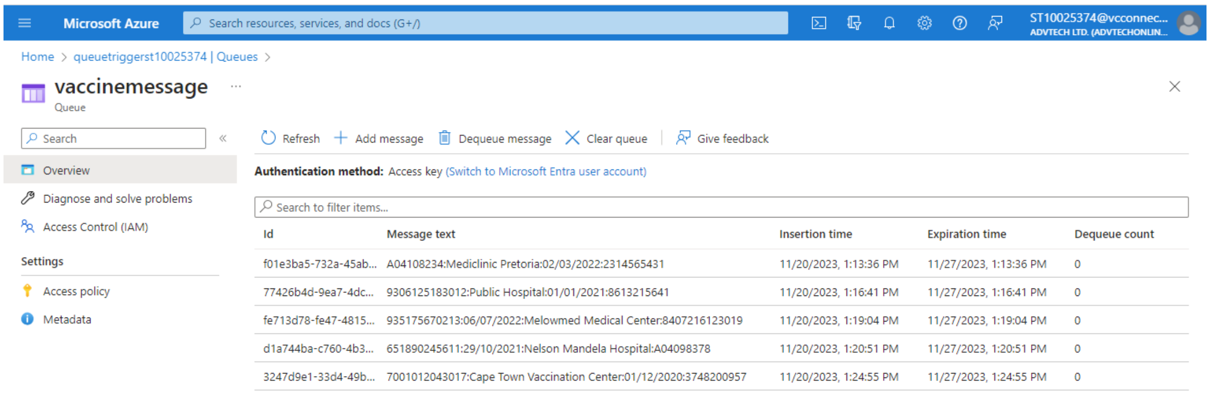


```csharp
                await containerClient.CreateIfNotExistsAsync();

                // Retrieve a reference to a BlobClient object. This object represents a specific blob (identified by 'blobName') in the container.
                BlobClient blobClient = containerClient.GetBlobClient(blobName);

                // Convert the message to a stream for blob upload
                byte[] byteArray = System.Text.Encoding.UTF8.GetBytes(myQueueItem);

                // Create a new memory stream from the byte array. Streams are used for blob data transfer operations.
                using var stream = new MemoryStream(byteArray);

                // Upload the stream data (converted queue message) to the blob in Azure Blob Storage.
                // If a blob with the same name already exists, it will be overwritten due to the 'overwrite: true' parameter.
                await blobClient.UploadAsync(stream, overwrite: true);

                log.LogInformation("Data saved to Azure Blob Storage.");
            }
            catch (Exception ex)
            {
                log.LogError($"Error: {ex.Message}");
            }
        }
    }
}
//-------------------------------------------------------< END >-------------------------------------------------------------//
```



```json
{
    "IsEncrypted": false,
  "Values": {
    "AzureWebJobsStorage": "DefaultEndpointsProtocol=https;AccountName=queuetriggerst10025374;AccountKey=/klJ15C9P/Uq47Rg3EFhzE4kc1QkRxXOVraYPPZECAoVK2TB/K8tOw
    "FUNCTIONS_WORKER_RUNTIME": "dotnet"
  }
}
```

Messages in the queue in Azure portal

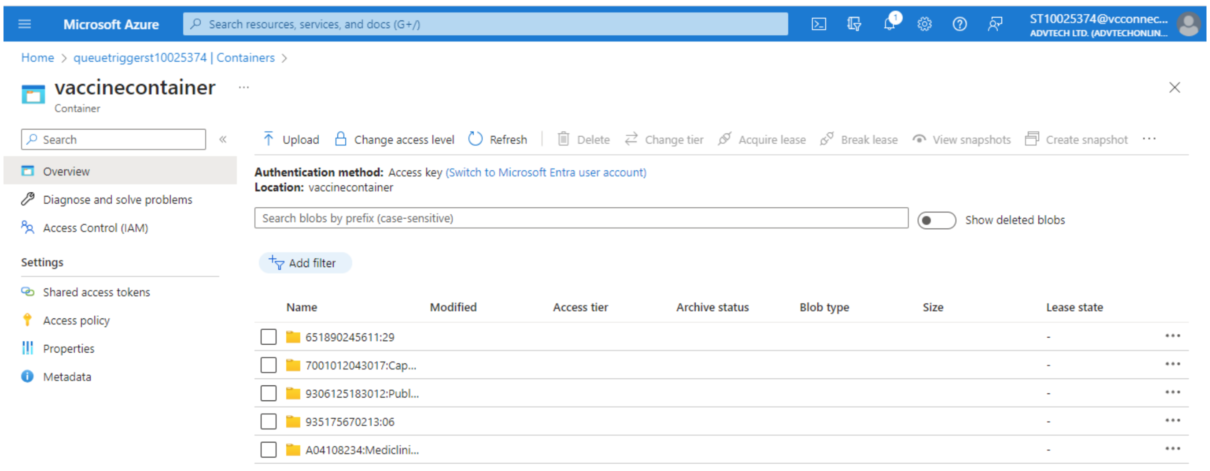

Home > queuetriggerst10025374 | Queues >

vaccinemessage
Queue

Authentication method: Access key (Switch to Microsoft Entra user account)

Id	Message text	Insertion time	Expiration time	Dequeue count
f01e3ba5-732a-45ab...	A04108234:Mediclinic Pretoria:02/03/2022:2314565431	11/20/2023, 1:13:36 PM	11/27/2023, 1:13:36 PM	0
77426b4d-9ea7-4dc...	9306125183012:Public Hospital:01/01/2021:8613215641	11/20/2023, 1:16:41 PM	11/27/2023, 1:16:41 PM	0
fe713d78-fe47-4815...	935175670213:06/07/2022:Melowmed Medical Center:8407216123019	11/20/2023, 1:19:04 PM	11/27/2023, 1:19:04 PM	0
d1a744ba-c760-4b3...	651890245611:29/10/2021:Nelson Mandela Hospital:A04098378	11/20/2023, 1:20:51 PM	11/27/2023, 1:20:51 PM	0
3247d9e1-33d4-49b...	7001012043017:Cape Town Vaccination Center:01/12/2020:3748200957	11/20/2023, 1:24:55 PM	11/27/2023, 1:24:55 PM	0

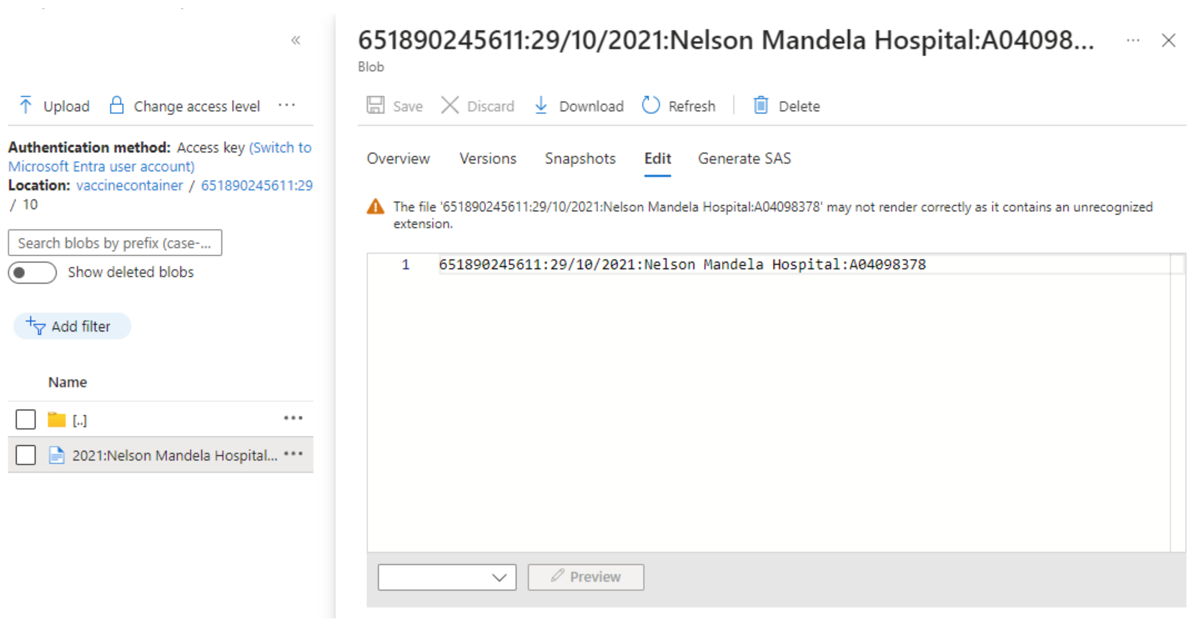
Corresponding messages in the changed component.


Home > queuetriggerst10025374 | Containers >

vaccinecontainer
Container

Authentication method: Access key (Switch to Microsoft Entra user account)
Location: vaccinecontainer

Name	Modified	Access tier	Archive status	Blob type	Size	Lease state
651890245611:29						-
7001012043017:Cap...						-
9306125183012:Publ...						-
935175670213:06						-
A04108234:Mediclini...						-

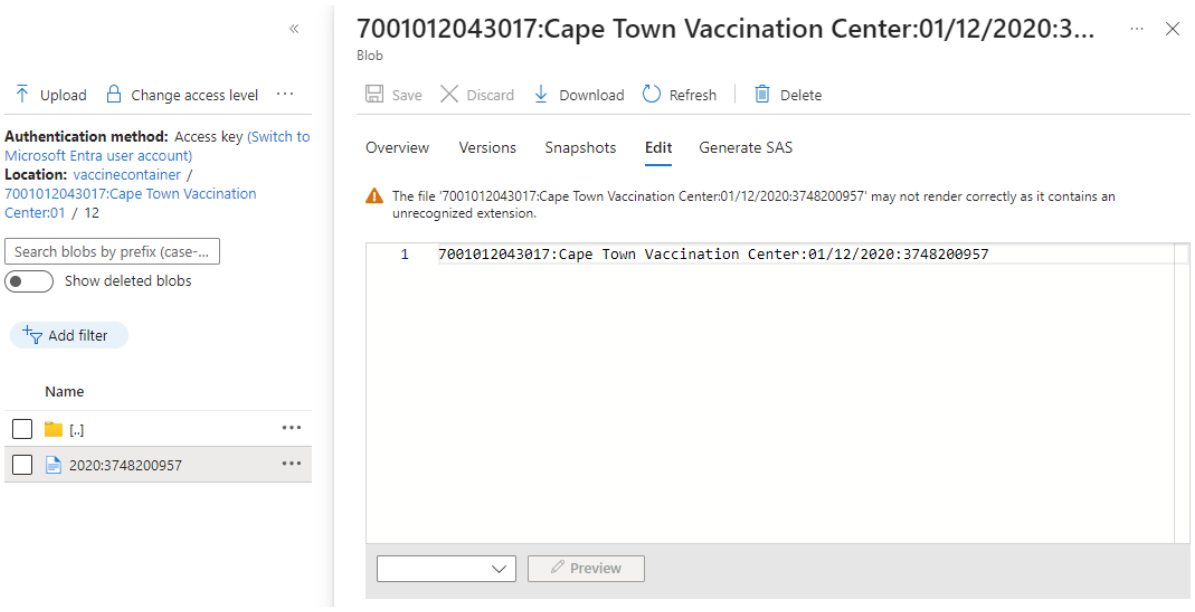

Authentication method: Access key (Switch to Microsoft Entra user account)
Location: vaccinecontainer / 651890245611:29 / 10

Name
[..]
2021:Nelson Mandela Hospital...

651890245611:29/10/2021:Nelson Mandela Hospital:A04098...
Blob

The file '651890245611:29/10/2021:Nelson Mandela Hospital:A04098378' may not render correctly as it contains an unrecognized extension.

```
651890245611:29/10/2021:Nelson Mandela Hospital:A04098378
```


Upload Change access level

Authentication method: Access key (Switch to Microsoft Entra user account)
Location: vaccinecontainer / 7001012043017:Cape Town Vaccination Center:01 / 12

Search blobs by prefix (case-...

Show deleted blobs

Add filter

Name

[..]

2020:3748200957

7001012043017:Cape Town Vaccination Center:01/12/2020:3...

Blob

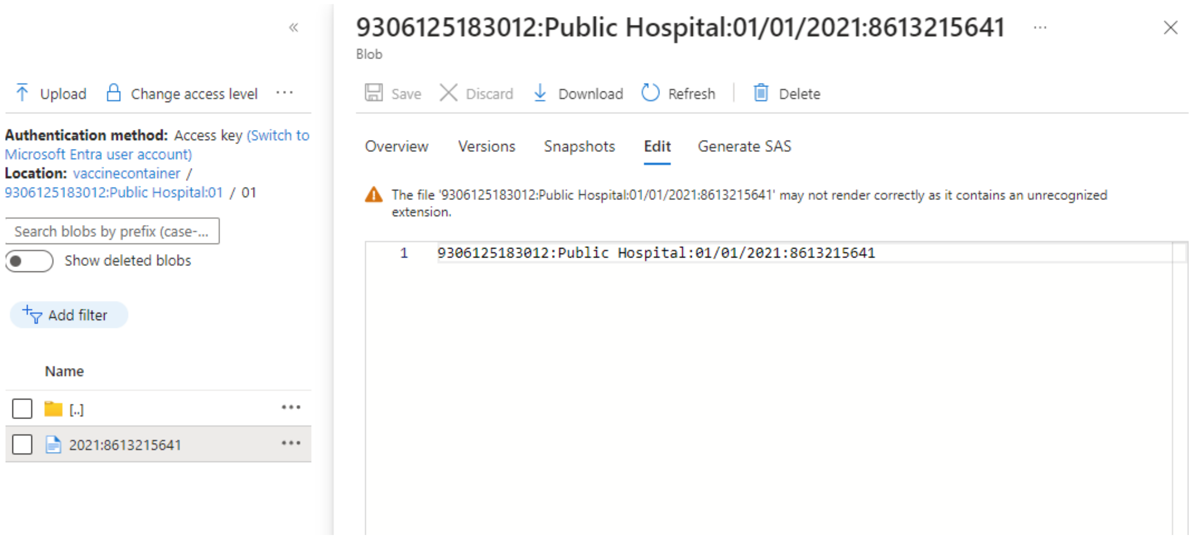
Save Discard Download Refresh Delete

Overview Versions Snapshots Edit Generate SAS

The file '7001012043017:Cape Town Vaccination Center:01/12/2020:3748200957' may not render correctly as it contains an unrecognized extension.

```
7001012043017:Cape Town Vaccination Center:01/12/2020:3748200957
```

Preview


Upload Change access level

Authentication method: Access key (Switch to Microsoft Entra user account)
Location: vaccinecontainer / 9306125183012:Public Hospital:01 / 01

Search blobs by prefix (case-...

Show deleted blobs

Add filter

Name

[..]

2021:8613215641

9306125183012:Public Hospital:01/01/2021:8613215641

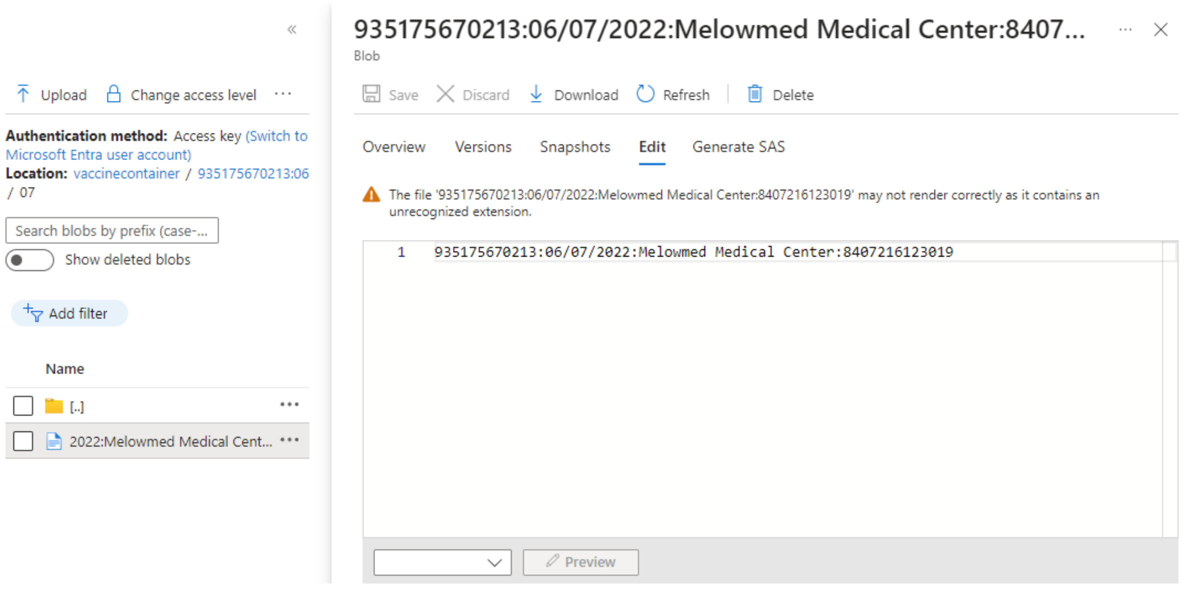
Blob

Save Discard Download Refresh Delete

Overview Versions Snapshots Edit Generate SAS

The file '9306125183012:Public Hospital:01/01/2021:8613215641' may not render correctly as it contains an unrecognized extension.

```
9306125183012:Public Hospital:01/01/2021:8613215641
```


Upload Change access level

Authentication method: Access key (Switch to Microsoft Entra user account)
Location: vaccinecontainer / 935175670213:06 / 07

Search blobs by prefix (case-...

Show deleted blobs

Add filter

Name

[..]

2022:Melowmed Medical Cent...

935175670213:06/07/2022:Melowmed Medical Center:8407...

Blob

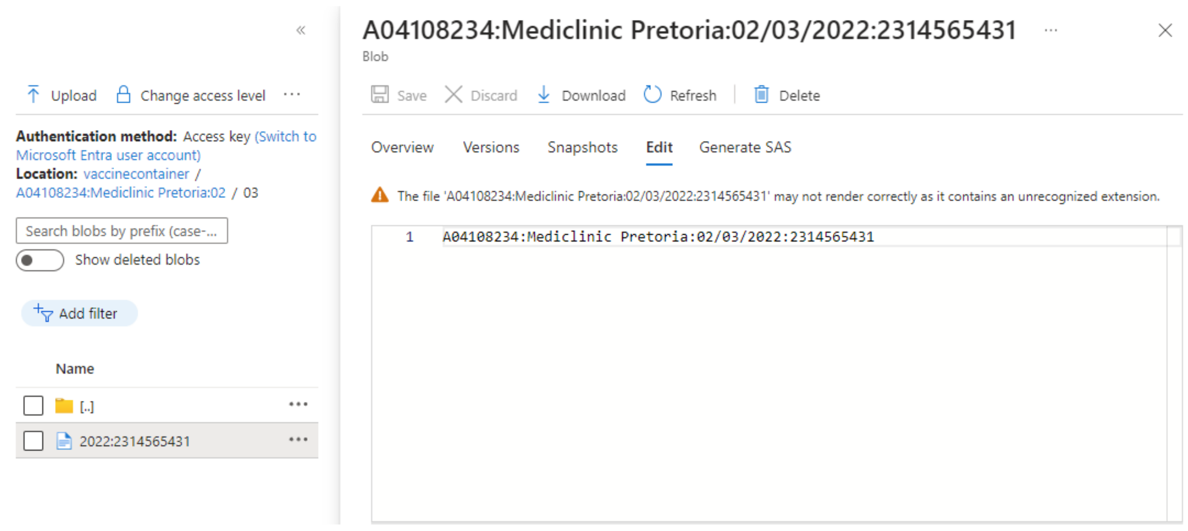
Save Discard Download Refresh Delete

Overview Versions Snapshots Edit Generate SAS

The file '935175670213:06/07/2022:Melowmed Medical Center:8407216123019' may not render correctly as it contains an unrecognized extension.

```
935175670213:06/07/2022:Melowmed Medical Center:8407216123019
```

Preview


A04108234:Mediclinic Pretoria:02/03/2022:2314565431

Blob

Authentication method: Access key (Switch to Microsoft Entra user account)
Location: vaccinecontainer / A04108234:Mediclinic Pretoria:02 / 03

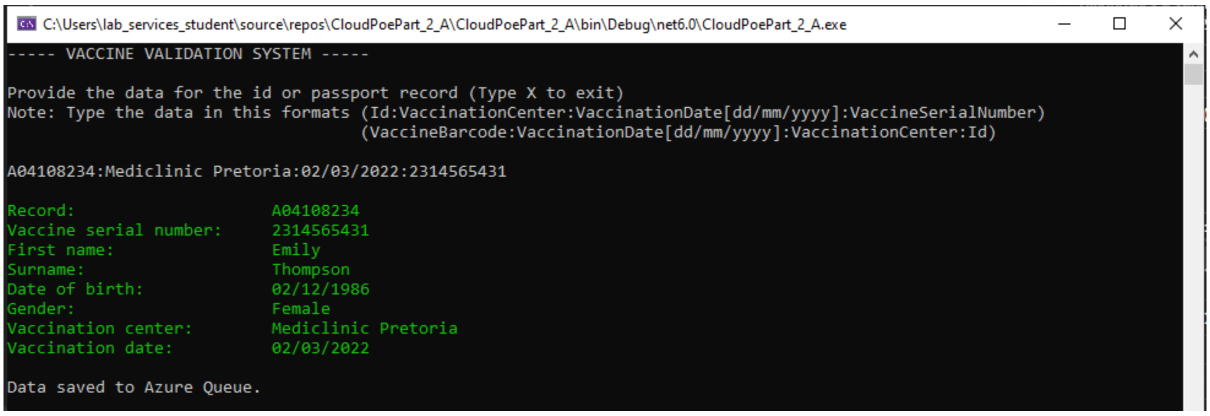
2022:2314565431

The file 'A04108234:Mediclinic Pretoria:02/03/2022:2314565431' may not render correctly as it contains an unrecognized extension.

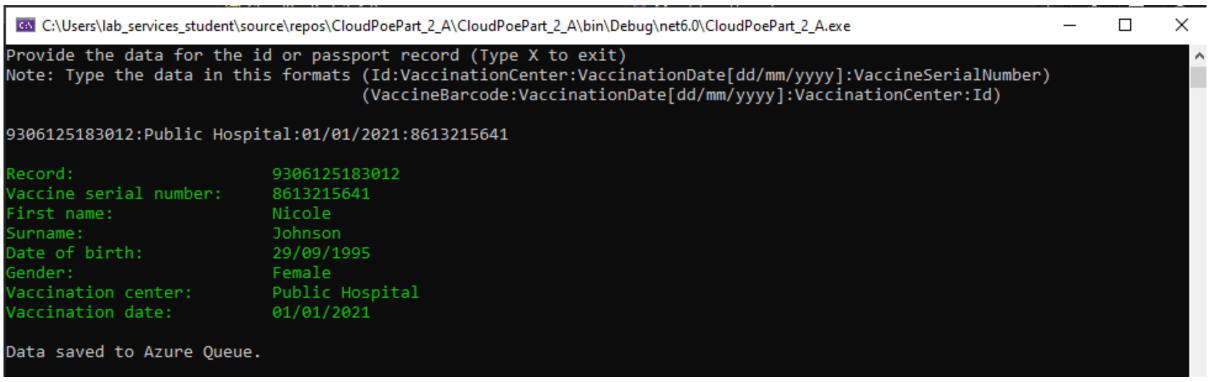
```
A04108234:Mediclinic Pretoria:02/03/2022:2314565431
```

Part D

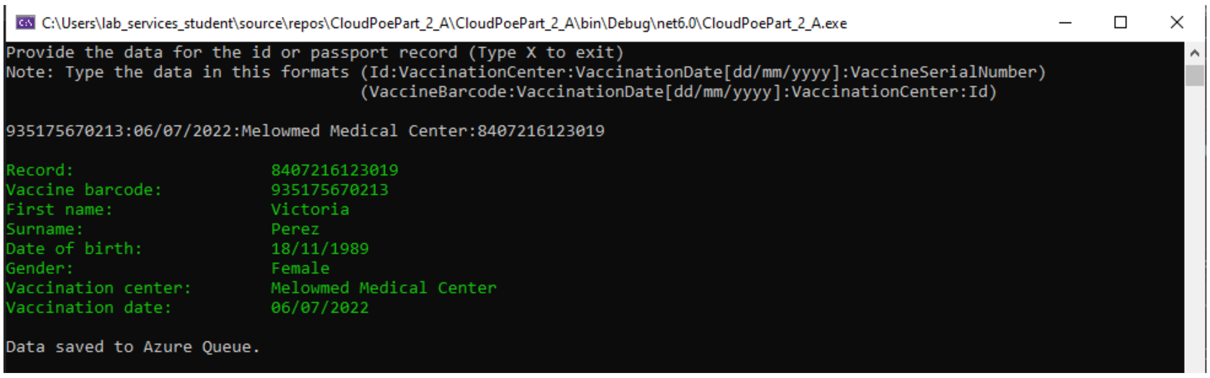
Screenshots of the console application running


C:\Users\lab_services_student\source\repos\CloudPoePart_2_A\CloudPoePart_2_A\bin\Debug\net6.0\CloudPoePart_2_A.exe

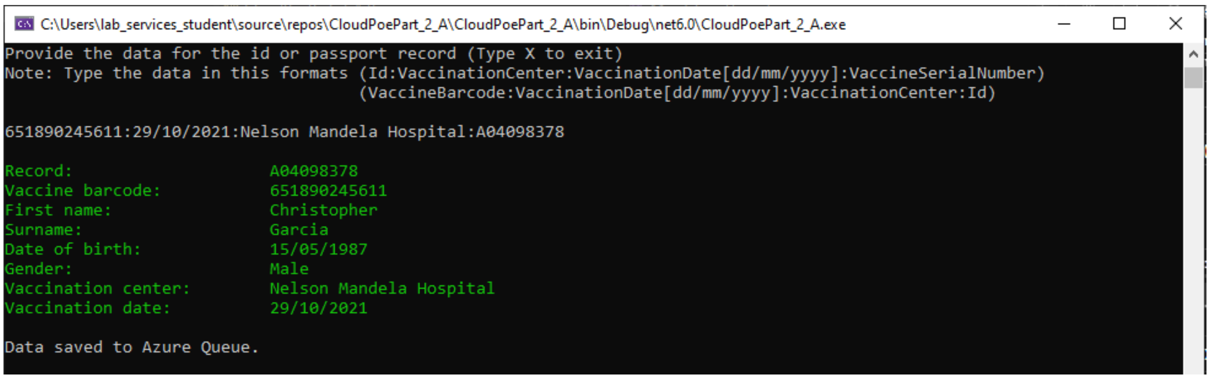
```
----- VACCINE VALIDATION SYSTEM -----

Provide the data for the id or passport record (Type X to exit)
Note: Type the data in this formats (Id:VaccinationCenter:VaccinationDate[dd/mm/yyyy]:VaccineSerialNumber)
                                    (VaccineBarcode:VaccinationDate[dd/mm/yyyy]:VaccinationCenter:Id)

A04108234:Mediclinic Pretoria:02/03/2022:2314565431

Record:                    A04108234
Vaccine serial number:     2314565431
First name:                Emily
Surname:                   Thompson
Date of birth:             02/12/1986
Gender:                    Female
Vaccination center:        Mediclinic Pretoria
Vaccination date:          02/03/2022

Data saved to Azure Queue.
```


C:\Users\lab_services_student\source\repos\CloudPoePart_2_A\CloudPoePart_2_A\bin\Debug\net6.0\CloudPoePart_2_A.exe

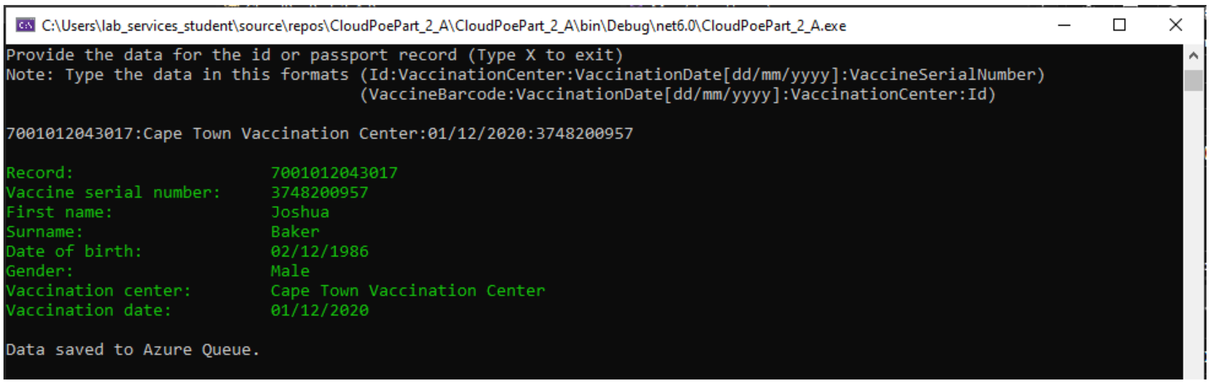
```
Provide the data for the id or passport record (Type X to exit)
Note: Type the data in this formats (Id:VaccinationCenter:VaccinationDate[dd/mm/yyyy]:VaccineSerialNumber)
                                    (VaccineBarcode:VaccinationDate[dd/mm/yyyy]:VaccinationCenter:Id)

9306125183012:Public Hospital:01/01/2021:8613215641

Record:                    9306125183012
Vaccine serial number:     8613215641
First name:                Nicole
Surname:                   Johnson
Date of birth:             29/09/1995
Gender:                    Female
Vaccination center:        Public Hospital
Vaccination date:          01/01/2021

Data saved to Azure Queue.
```


C:\Users\lab_services_student\source\repos\CloudPoePart_2_A\CloudPoePart_2_A\bin\Debug\net6.0\CloudPoePart_2_A.exe

```
Provide the data for the id or passport record (Type X to exit)
Note: Type the data in this formats (Id:VaccinationCenter:VaccinationDate[dd/mm/yyyy]:VaccineSerialNumber)
                                    (VaccineBarcode:VaccinationDate[dd/mm/yyyy]:VaccinationCenter:Id)

935175670213:06/07/2022:Melowmed Medical Center:8407216123019

Record:                    8407216123019
Vaccine barcode:           935175670213
First name:                Victoria
Surname:                   Perez
Date of birth:             18/11/1989
Gender:                    Female
Vaccination center:        Melowmed Medical Center
Vaccination date:          06/07/2022

Data saved to Azure Queue.
```

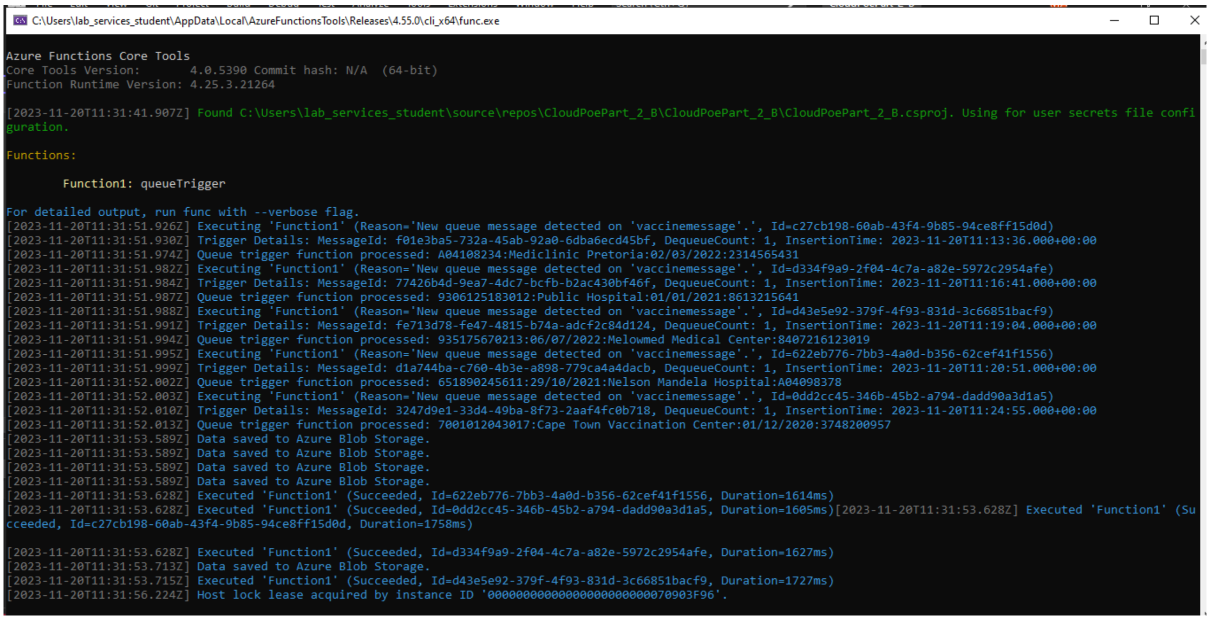


```
C:\Users\lab_services_student\source\repos\CloudPoePart_2_A\CloudPoePart_2_A\bin\Debug\net6.0\CloudPoePart_2_A.exe

Provide the data for the id or passport record (Type X to exit)
Note: Type the data in this formats (Id:VaccinationCenter:VaccinationDate[dd/mm/yyyy]:VaccineSerialNumber)
                                    (VaccineBarcode:VaccinationDate[dd/mm/yyyy]:VaccinationCenter:Id)

651890245611:29/10/2021:Nelson Mandela Hospital:A04098378

Record:                    A04098378
Vaccine barcode:           651890245611
First name:                Christopher
Surname:                   Garcia
Date of birth:             15/05/1987
Gender:                    Male
Vaccination center:        Nelson Mandela Hospital
Vaccination date:          29/10/2021

Data saved to Azure Queue.
```



```
C:\Users\lab_services_student\source\repos\CloudPoePart_2_A\CloudPoePart_2_A\bin\Debug\net6.0\CloudPoePart_2_A.exe

Provide the data for the id or passport record (Type X to exit)
Note: Type the data in this formats (Id:VaccinationCenter:VaccinationDate[dd/mm/yyyy]:VaccineSerialNumber)
                                    (VaccineBarcode:VaccinationDate[dd/mm/yyyy]:VaccinationCenter:Id)

7001012043017:Cape Town Vaccination Center:01/12/2020:3748200957

Record:                    7001012043017
Vaccine serial number:     3748200957
First name:                Joshua
Surname:                   Baker
Date of birth:             02/12/1986
Gender:                    Male
Vaccination center:        Cape Town Vaccination Center
Vaccination date:          01/12/2020

Data saved to Azure Queue.
```

Screenshot of Queue Trigger running



```
C:\Users\lab_services_student\AppData\Local\AzureFunctionsTools\Releases\4.55.0\cli_x64\func.exe

Azure Functions Core Tools
Core Tools Version:       4.0.5390 Commit hash: N/A  (64-bit)
Function Runtime Version: 4.25.3.21264

[2023-11-20T11:31:41.907Z] Found C:\Users\lab_services_student\source\repos\CloudPoePart_2_B\CloudPoePart_2_B\CloudPoePart_2_B.csproj. Using for user secrets file confi
guration.

Functions:

        Function1: queueTrigger

For detailed output, run func with --verbose flag.
[2023-11-20T11:31:51.926Z] Executing 'Function1' (Reason='New queue message detected on 'vaccinemessage'.', Id=c27cb198-60ab-43f4-9b85-94ce8ff15d0d)
[2023-11-20T11:31:51.930Z] Trigger Details: MessageId: f01e3ba5-732a-45ab-92a0-6dba6ecd45bf, DequeueCount: 1, InsertionTime: 2023-11-20T11:13:36.000+00:00
[2023-11-20T11:31:51.974Z] Queue trigger function processed: A04108234:Mediclinic Pretoria:02/03/2022:2314565431
[2023-11-20T11:31:51.982Z] Executing 'Function1' (Reason='New queue message detected on 'vaccinemessage'.', Id=d334f9a9-2f04-4c7a-a82e-5972c2954afe)
[2023-11-20T11:31:51.984Z] Trigger Details: MessageId: 77426b4d-9ea7-4dc7-bcfb-b2ac430bf46f, DequeueCount: 1, InsertionTime: 2023-11-20T11:16:41.000+00:00
[2023-11-20T11:31:51.987Z] Queue trigger function processed: 9306125183012:Public Hospital:01/01/2021:8613215641
[2023-11-20T11:31:51.988Z] Executing 'Function1' (Reason='New queue message detected on 'vaccinemessage'.', Id=d43e5e92-379f-4f93-831d-3c66851bacf9)
[2023-11-20T11:31:51.991Z] Trigger Details: MessageId: fe713d78-fe47-4815-b74a-adcf2c84d124, DequeueCount: 1, InsertionTime: 2023-11-20T11:19:04.000+00:00
[2023-11-20T11:31:51.994Z] Queue trigger function processed: 935175670213:06/07/2022:Melowmed Medical Center:8407216123019
[2023-11-20T11:31:51.995Z] Executing 'Function1' (Reason='New queue message detected on 'vaccinemessage'.', Id=622eb776-7bb3-4a0d-b356-62cef41f1556)
[2023-11-20T11:31:51.999Z] Trigger Details: MessageId: d1a744ba-c760-4b3e-a898-779ca4a4dacb, DequeueCount: 1, InsertionTime: 2023-11-20T11:20:51.000+00:00
[2023-11-20T11:31:52.002Z] Queue trigger function processed: 651890245611:29/10/2021:Nelson Mandela Hospital:A04098378
[2023-11-20T11:31:52.003Z] Executing 'Function1' (Reason='New queue message detected on 'vaccinemessage'.', Id=0dd2cc45-346b-45b2-a794-dadd90a3d1a5)
[2023-11-20T11:31:52.010Z] Trigger Details: MessageId: 3247d9e1-33d4-49ba-8f73-2aaf4fc0b718, DequeueCount: 1, InsertionTime: 2023-11-20T11:24:55.000+00:00
[2023-11-20T11:31:52.013Z] Queue trigger function processed: 7001012043017:Cape Town Vaccination Center:01/12/2020:3748200957
[2023-11-20T11:31:53.589Z] Data saved to Azure Blob Storage.
[2023-11-20T11:31:53.589Z] Data saved to Azure Blob Storage.
[2023-11-20T11:31:53.589Z] Data saved to Azure Blob Storage.
[2023-11-20T11:31:53.589Z] Data saved to Azure Blob Storage.
[2023-11-20T11:31:53.628Z] Executed 'Function1' (Succeeded, Id=622eb776-7bb3-4a0d-b356-62cef41f1556, Duration=1614ms)
[2023-11-20T11:31:53.628Z] Executed 'Function1' (Succeeded, Id=0dd2cc45-346b-45b2-a794-dadd90a3d1a5, Duration=1605ms)[2023-11-20T11:31:53.628Z] Executed 'Function1' (Su
cceeded, Id=c27cb198-60ab-43f4-9b85-94ce8ff15d0d, Duration=1758ms)

[2023-11-20T11:31:53.628Z] Executed 'Function1' (Succeeded, Id=d334f9a9-2f04-4c7a-a82e-5972c2954afe, Duration=1627ms)
[2023-11-20T11:31:53.713Z] Data saved to Azure Blob Storage.
[2023-11-20T11:31:53.715Z] Executed 'Function1' (Succeeded, Id=d43e5e92-379f-4f93-831d-3c66851bacf9, Duration=1727ms)
[2023-11-20T11:31:56.224Z] Host lock lease acquired by instance ID '000000000000000000000000070903F96'.
```