IN PARTNERSHIP WITH PLYMOUTH UNIVERSITY

Name: V P N Sulakshika
Student Reference Number: 10707385

Module Code: PUSL3111	Module Name: API Software Development
Coursework Title: Sri Lanka Bureau of Foreign Employment (SLBFE) - REST API Web and Mobile Applications	
Deadline Date: Wednesday, 11 May 2022, 11:30 AM	Member of staff responsible for coursework: Dr. Rasika Ranaweera
Programme: BSc (Hons) Software Engineering	

Please note that University Academic Regulations are available under Rules and Regulations on the University website www.plymouth.ac.uk/studenthandbook.

Group work: please list all names of all participants formally associated with this work and state whether the work was undertaken alone or as part of a team. Please note you may be required to identify individual responsibility for component parts.

Group No - 21

10707385 - V P N Sulakshika **(Group Leader)**

10707241 - H S Kaushalya

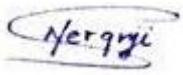
10707417 – H J K I Wijerama

10707129 – K L D Anupama

10707372 - P D S Sigera

10707291 – S S N S Nevins

We confirm that we have read and understood the Plymouth University regulations relating to Assessment Offences and that we are aware of the possible penalties for any breach of these regulations. We confirm that this is the independent work of the group.

Signed on behalf of the group: 

Individual assignment: ***I confirm that I have read and understood the Plymouth University regulations relating to Assessment Offences and that I am aware of the possible penalties for any breach of these regulations. I confirm that this is my own independent work.***

Signed :

Use of translation software: failure to declare that translation software or a similar writing aid has been used will be treated as an assessment offence.

I *have used/not used translation software.

If used, please state name of software……

Overall mark _____% Assessors Initials _____ Date_____

*Please delete as appropriate Sci/ps/d:/students/cwkfrontcover/2013/14

Table of Contents

List of Figures

Appendix

In this API coursework we have developed the web application as well as the mobile application. In the web application we have developed the backend and frontend. The backend was implemented in visual studio community 2022and the frontend was implemented in visual studio code 2019. The backend was developed with Swagger and for the frontend was developed with ReactJS. In the mobile application the frontend was developed in JAVA and the backend was developed by .NET framework. The APIs is connected to the frontend through Retrofit.

The web API is developed in ReactJS. The interfaces use the REST architecture, and it delivers the data in both XML and JSON formats. These are the functions that we have used POST, PUT GET, DELETE. In the POST method both the citizens and the officers can register themselves using the registration form that we have developed. Registration form includes NIC, name age, address, current location, profession, email, affiliation, password. In PUT method it is the updating of the server side so the job seeker can update their qualifications and upload the certificates also officers can verify their own information. In the GET method it is requestion the server to send the required data that is requested so that the officers should be able to access the citizens personal information by using the NIC also the company officers can be able to find the applicant based on their qualifications. The DELETE method is basically deleting a personal account if the person has passed away.

1. Introduction

An American computer programmer and technology executive once said that “If the web can be evolved to include the missing APIs and have better performance, [developers] won’t need to go beyond the web. – Author: Brendan Eich” (Quotestats, n.d.) It means that APIs are the best tools that can be used when developing an application. A web and a mobile application were built for the Sri Lanka Bureau of Foreign Employment (SLBFE) to cater with their customer more smoothly and accurately. So, these applications are built with the help of APIs. In this system both the mobile and web has the same functionalities, so the mode of access is the only differentiation here. There are three users in this application they are: Citizens, Bureau Officer, and the Admin. All the users must login with the system and the officer and citizen can also sign up with the system. Citizens can edit their profile details, upload any documents that they want and make a complaint. The bureau can view all the citizens and select them according to their preferred way. Admin can view all the citizens and officers and reply to complaints and delete any profiles which are no longer in use.

The technologies used to build these applications are: For the web application the frontend was created with React JS while the backend was created with ASP.NET. And in the mobile application the frontend was created with Java language and the web application backend was connected to it. The IDE used to develop the mobile application is Android Studio and to connect the front end and the APIs Retrofit 2 was used. The database used here is MySQL and the APIs are tested via Shagger and Postman tools. And all these APIs are working with the designed user interfaces smoothly by functioning well as expected. These tools and technologies will be discussed briefly in relevant topics.

2. API Documentation

2.1. List of APIs

A. /Auth/GetAdmin

- The SLBFE staff (Admin) can login to the system with details including the UserName, and the Password.

B. /Auth/SignUp/Create

- Citizens and Officers can register themselves with details including a national ID, name, age, address, profession, email, affiliation, password, and qualification.

C. /Auth/SignIn

- Citizens and Officers can login to the system with details including an email, affiliation, and password.

D. /Auth/ReadCitizenInformation

- The Bureau Officers must be able to see information provided by the job seekers.

E. /Auth/CitizenInformationByNIC/Get/{nic}

- Officers should be able to access any citizen's information by their national id.

F. /Auth/CitizenInformationByQualification/Get/{qualification}

- Company officers should be able to find candidates based on qualifications.

G. /Auth/UpdateCitizenInformation/UpdateQualification/{nic}

- Job seekers should be able to update their qualifications by NIC.

H. /Auth/UploadCitizenDocuments

- Job seekers should be able to upload certificates.

I. /Auth/DeleteCitizen/Delete/{userId}

- The SLBFE staff can deactivate an individual's account if the citizen is deceased.

J. /Auth/ReadComplaints

- The Bureau Officers should be able to see the content.

K. /Auth/CreateComplaint

- Any citizen can make a complaint.

L. /Auth/UpdateComplaintInformation/Update/{complaintId}

- The Bureau Officers should be able to reply accordingly.

2.2. Screenshots

2.2.1. Code Screenshots - Backend

2.2.1.2. Web Application

Admin Sign-In

- The below two screenshots represent the Admin Sign-In.

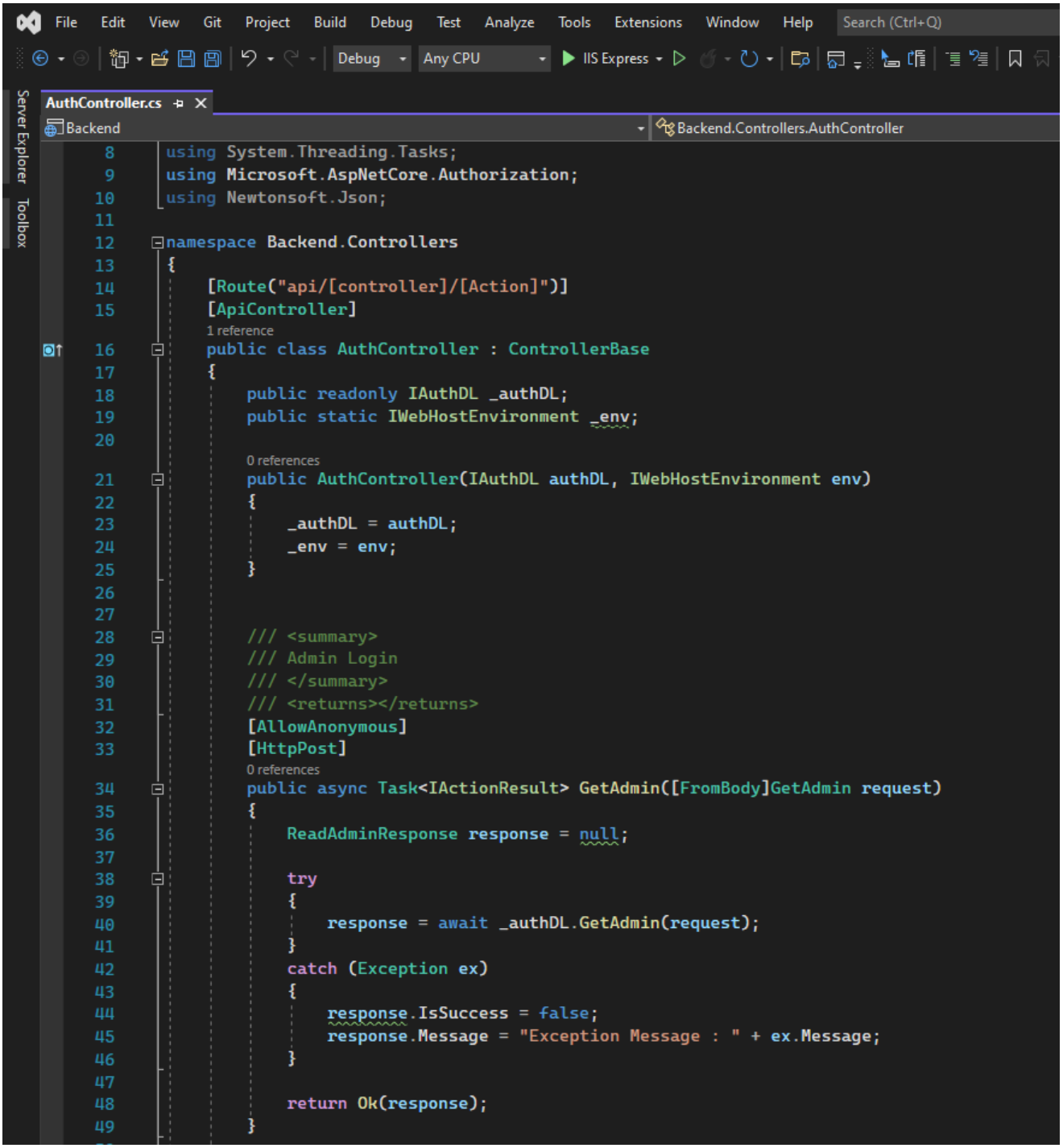


Figure 1 - Admin Sign-In method in the AuthController.cs

```
using ExcelDataReader;
using System.Data;
using System.Text.RegularExpressions;

namespace Backend.DataAccessLayer
{
    2 references
    public class AuthDL : IAuthDL
    {
        public readonly IConfiguration _configuration;
        public readonly MySqlConnection _mySqlConnection;
        public readonly string EmailRegex = @"^[0-9a-zA-Z]+([._+-][0-9a-zA-Z]+)*@[0-9a-zA-Z]+.[a-zA-Z]{2,4}([.][a-zA-Z]{2,3})?$";

        int ConnectionTimeOut = 180;

        0 references
        public AuthDL(IConfiguration configuration)
        {
            _configuration = configuration;
            _mySqlConnection = new MySqlConnection(_configuration[key: "ConnectionStrings:MySqlDBConnectionString"]);
        }


        /// <summary>
        /// Admin Login
        /// </summary>
        /// <param name="request"></param>
        /// <returns></returns>
        3 references
        public async Task<ReadAdminResponse> GetAdmin(GetAdmin request)
        {
            ReadAdminResponse response = new ReadAdminResponse();
            response.readAdmin = new List<GetAdmin>();

            response.IsSuccess = true;
            response.Message = "Successful";

            try
            {
                if (_mySqlConnection.State != System.Data.ConnectionState.Open)
                {
                    await _mySqlConnection.OpenAsync();
                }

                //Check Null Or Empty
                if (string.IsNullOrEmpty(request.UserName) || string.IsNullOrEmpty(request.Password))
                {
                    response.Message = "Username Or Password cannot Null or Empty";
                }

                //SQL Query
                string SqlQuery = @"SELECT *
                                    FROM slbfe.admin
                                    WHERE UserName=@UserName AND Password=@Password;";

                using (MySqlCommand sqlCommand = new MySqlCommand(SqlQuery, _mySqlConnection))
                {
                    sqlCommand.CommandType = System.Data.CommandType.Text;
                    sqlCommand.CommandTimeout = 180;
                    sqlCommand.Parameters.AddWithValue("@UserName", request.UserName);
                    sqlCommand.Parameters.AddWithValue("@Password", request.Password);

                    using (DbDataReader dataReader = await sqlCommand.ExecuteReaderAsync())
                    {
                        if (dataReader.HasRows)
                        {
                            response.Message = "Login Successfull";
                        }
                        else
                        {
                            response.IsSuccess = false;
                            response.Message = "Login Failed";
                            return response;
                        }
                    }
                }
                catch (Exception ex)
                {
                    response.IsSuccess = false;
                    response.Message = ex.Message;
                }
                finally
                {
                    await _mySqlConnection.CloseAsync();
                    await _mySqlConnection.DisposeAsync();
                }

                return response;
            }
```

Figure 2 - Admin Sign-In method in the AuthDL.cs

Citizen and Officer Sign-Up

- The below two screenshots represent the Citizen and Officer Sign-Up.

```
/// <summary>
/// Sign Up
/// </summary>
/// <param name="request"></param>
/// <returns></returns>
[AllowAnonymous]
[HttpPost]
[Route("Create")]
0 references
public async Task<ActionResult> SignUp([FromBody] SignUpRequest request)
{
    SignUpResponse response = new SignUpResponse();
    try
    {
        response = await _authDL.SignUp(request);
    }
    catch(Exception ex)
    {
        response.IsSuccess = false;
        response.Message = "Exception Message : " + ex.Message;
    }

    return Ok(response);
}
```

Figure 3 - Citizen & Officer: Sign-Up method in the AuthController.cs

```
/// <summary>
/// Sign Up
/// </summary>
/// <param name="request"></param>
/// <returns></returns>
2 references
public async Task<SignUpResponse> SignUp(SignUpRequest request)
{
    SignUpResponse response = new SignUpResponse();

    response.IsSuccess = true;
    response.Message = "Successful";

    try
    {
        if (_mySqlConnection.State != System.Data.ConnectionState.Open)
        {
            await _mySqlConnection.OpenAsync();
        }

        //Password Validation
        if (!request.Password.Equals(request.ConfigPassword))
        {
            response.IsSuccess = false;
            response.Message = "Password & Confirm Password not Match";
            return response;
        }

        //Email Validation
        if (!(Regex.IsMatch(request.Email, EmailRegex)))
        {
            response.IsSuccess = false;
            response.Message = "Email is not correct format";
            return response;
        }

        //SQL Query
        string SqlQuery = @"INSERT INTO
                            slbfe.user_details
                            (NIC, Name, Address, Age, Profession, Email, Password, Affiliation, Qualification) VALUES
                            (@NIC, @Name, @Address, @Age, @Profession, @Email, @Password, @Affiliation, @Qualification)";

        using (MySqlCommand sqlCommand = new MySqlCommand(SqlQuery, _mySqlConnection))
        {
            sqlCommand.CommandType = System.Data.CommandType.Text;
            sqlCommand.CommandTimeout = 180;
            sqlCommand.Parameters.AddWithValue("@NIC", request.NIC);
            sqlCommand.Parameters.AddWithValue("@Name", request.Name);
            sqlCommand.Parameters.AddWithValue("@Address", request.Address);
            sqlCommand.Parameters.AddWithValue("@Age", request.Age);
            sqlCommand.Parameters.AddWithValue("@Profession", request.Profession);
            sqlCommand.Parameters.AddWithValue("@Email", request.Email);
            sqlCommand.Parameters.AddWithValue("@Password", request.Password);
            sqlCommand.Parameters.AddWithValue("@Affiliation", request.Affiliation);
            sqlCommand.Parameters.AddWithValue("@Qualification", request.Qualification);
            int Status = await sqlCommand.ExecuteNonQueryAsync();

            if(Status <= 0)
            {
                response.IsSuccess = false;
                response.Message = "Something Went Wrong";
                return response;
            }
        }
    }
    catch (Exception ex)
    {
        response.IsSuccess = false;
        response.Message = ex.Message;
    }
    finally
    {
        await _mySqlConnection.CloseAsync();
        await _mySqlConnection.DisposeAsync();
    }

    return response;
}
```

Figure 4- Citizen & Officer: Sign-Up method in the AuthDL.cs

Citizen and Officer Sign-In

- The below two screenshots represent the Citizen and Officer Sign-In.

```
/// <summary>
/// Sign In
/// </summary>
/// <param name="request"></param>
/// <returns></returns>
[AllowAnonymous]
[HttpPost]
0 references
public async Task<ActionResult> SignIn([FromBody] SignInRequest request)
{
    SignInResponse response = new SignInResponse();
    try
    {
        response = await _authDL.SignIn(request);
    }
    catch (Exception ex)
    {
        response.IsSuccess = false;
        response.Message = "Exception Message : " + ex.Message;
    }

    return Ok(response);
}
```

Figure 5 - Citizen & Officer: Sign-In in the AuthController.cs

```
/// <summary>
/// Sign In
/// </summary>
/// <param name="request"></param>
/// <returns></returns>
2 references
public async Task<SignInResponse> SignIn(SignInRequest request)
{
    SignInResponse response = new SignInResponse();

    response.IsSuccess = true;
    response.Message = "Successful";

    try
    {
        if(_mySqlConnection.State != System.Data.ConnectionState.Open)
        {
            await _mySqlConnection.OpenAsync();
        }

        //Check Null Or Empty
        if (string.IsNullOrEmpty(request.Email) || string.IsNullOrEmpty(request.Password) || string.IsNullOrEmpty(request.Affiliation))
        {
            response.Message = "Usename Or Password Or Affiliation cannot Null or Empty";
        }

        //Email Validation
        if (!(Regex.IsMatch(request.Email, EmailRegex)))
        {
            response.IsSuccess = false;
            response.Message = "Email is not correct format!";
            return response;
        }

        //SQL Query
        string SqlQuery = @"SELECT *
                            FROM slbfe.user_details
                            WHERE Email=@Email AND Password=@Password AND Affiliation=@Affiliation;";

        using (MySqlCommand sqlCommand = new MySqlCommand(SqlQuery, _mySqlConnection))
        {
            sqlCommand.CommandType = System.Data.CommandType.Text;
            sqlCommand.CommandTimeout = 180;
            sqlCommand.Parameters.AddWithValue("@Email", request.Email);
            sqlCommand.Parameters.AddWithValue("@Password", request.Password);
            sqlCommand.Parameters.AddWithValue("@Affiliation", request.Affiliation);

            using (DbDataReader dataReader = await sqlCommand.ExecuteReaderAsync())
            {
                if (dataReader.HasRows)
                {
                    response.Message = "Login Successfull";
                }
                else
                {
                    response.IsSuccess = false;
                    response.Message = "Login Failed";
                    return response;
                }
            }
        }
    }
    catch(Exception ex)
    {
        response.IsSuccess = false;
        response.Message = ex.Message;
    }
    finally
    {
        await _mySqlConnection.CloseAsync();
        await _mySqlConnection.DisposeAsync();
    }

    return response;
}
```

Figure 6 - Citizen & Officer: Sign-In method in the AuthDL.cs

Get List of Citizen Information

- The below two screenshots represent the List of Citizen Information.

```
/// <summary>
/// Get Citizen Information List
/// </summary>
/// <returns></returns>
[AllowAnonymous]
[HttpGet]
0 references
public async Task<IActionResult> ReadCitizenInformation()
{
    ReadCitizenInformationResponse response = null;
    try
    {
        response = await _authDL.ReadCitizenInformation();
    }
    catch (Exception ex)
    {
        response.IsSuccess = false;
        response.Message = "Exception Message : " + ex.Message;
    }

    return Ok(response);
}
```

Figure 7 - Read Citizen Information method in the AuthController.cs

```
/// <summary>
/// Get Citizen Information List
/// </summary>
/// <returns></returns>
3 references
public async Task<ReadCitizenInformationResponse> ReadCitizenInformation()
{
    ReadCitizenInformationResponse response = new ReadCitizenInformationResponse();
    response.readCitizenInformation = new List<ReadCitizenInformation>();

    response.IsSuccess = true;
    response.Message = "Successful";

    try
    {
        //SQL Query
        string SqlQuery = @"SELECT * FROM slbfe.user_details WHERE Affiliation = 'Citizen' ";

        using (MySqlCommand sqlCommand = new MySqlCommand(SqlQuery, _mySqlConnection))
        {
            await _mySqlConnection.OpenAsync();

            using (DbDataReader _sqlDataReader = await sqlCommand.ExecuteReaderAsync())
            {
                if (_sqlDataReader.HasRows)
                {
                    while (await _sqlDataReader.ReadAsync())
                    {
                        ReadCitizenInformation getResponse = new ReadCitizenInformation();

                        getResponse.UserId = _sqlDataReader["UserId"] != DBNull.Value ? Convert.ToInt32(_sqlDataReader["UserId"]) : 0;

                        getResponse.NIC = _sqlDataReader["NIC"] != DBNull.Value ? _sqlDataReader["NIC"].ToString() : string.Empty;

                        getResponse.Name = _sqlDataReader["Name"] != DBNull.Value ? _sqlDataReader["Name"].ToString() : string.Empty;

                        getResponse.Address = _sqlDataReader["Address"] != DBNull.Value ? _sqlDataReader["Address"].ToString() : string.Empty;

                        getResponse.Age = _sqlDataReader["Age"] != DBNull.Value ? Convert.ToInt32(_sqlDataReader["Age"]) : 0;

                        getResponse.Profession = _sqlDataReader["Profession"] != DBNull.Value ? _sqlDataReader["Profession"].ToString() : string.Empty;

                        getResponse.Email = _sqlDataReader["Email"] != DBNull.Value ? _sqlDataReader["Email"].ToString() : string.Empty;

                        getResponse.Qualification = _sqlDataReader["Qualification"] != DBNull.Value ? _sqlDataReader["Qualification"].ToString() : string.Empty;

                        response.readCitizenInformation.Add(getResponse);
                    }
                }
                else
                {
                    response.Message = "No data Return";
                }
            }
        }
    }
    catch (Exception ex)
    {
        response.IsSuccess = false;
        response.Message = "Exception Message : " + ex.Message;
    }
    finally
    {
        await _mySqlConnection.CloseAsync();
        await _mySqlConnection.DisposeAsync();
    }

    return response;
}
```

Figure 8 - Read Citizen Information method in the AuthDL.cs

Get Citizen Information By NIC

- The below two screenshots represent the Citizen Information by their NIC.

```
/// <summary>
/// Get Citizen Information By NIC
/// </summary>
/// <param name="request"></param>
/// <returns></returns>
[AllowAnonymous]
[HttpGet]
[Route("Get/{NIC}")]
0 references
public async Task<IActionResult> CitizenInformationByNIC([FromHeader] CitizenInformationByNICRequest request)
{
    CitizenInformationByNICResponse response = null;
    try
    {
        response = await _authDL.GetCitizenInformationByNIC(request);
    }
    catch (Exception ex)
    {
        response.IsSuccess = false;
        response.Message = "Exception Message : " + ex.Message;
    }

    return Ok(response);
}
```

Figure 9 - Citizen Information by NIC method in the AuthController.cs

```
/// <summary>
/// Get Citizen Information By NIC
/// </summary>
/// <param name="request"></param>
/// <returns></returns>
3 references
public async Task<CitizenInformationByNICResponse> GetCitizenInformationByNIC(CitizenInformationByNICRequest request)
{
    CitizenInformationByNICResponse response = new CitizenInformationByNICResponse();

    response.IsSuccess = true;
    response.Message = "Successful";

    try
    {
        //SQL Query
        string SqlQuery = @"SELECT * FROM slbfe.user_details WHERE NIC = @NIC AND Affiliation = 'Citizen'";

        //Check Null Or Empty
        if (string.IsNullOrEmpty(request.NIC))
        {
            response.Message = "NIC cannot Null or Empty";
        }

        using (MySqlCommand sqlCommand = new MySqlCommand(SqlQuery, _mySqlConnection))
        {
            sqlCommand.CommandType = System.Data.CommandType.Text;
            sqlCommand.CommandTimeout = ConnectionTimeOut;
            sqlCommand.Parameters.AddWithValue("@NIC", request.NIC);
            await _mySqlConnection.OpenAsync();

            using (DbDataReader _sqlDataReader = await sqlCommand.ExecuteReaderAsync())
            {
                if (_sqlDataReader.HasRows)
                {
                    await _sqlDataReader.ReadAsync();
                    response.getCitizenInformationByNIC = new GetCitizenInformationByNIC();

                    response.getCitizenInformationByNIC.Name = _sqlDataReader["Name"] != DBNull.Value ? _sqlDataReader["Name"].ToString() : string.Empty;
                    response.getCitizenInformationByNIC.Address = _sqlDataReader["Address"] != DBNull.Value ? _sqlDataReader["Address"].ToString() : string.Empty;
                    response.getCitizenInformationByNIC.Age = _sqlDataReader["Age"] != DBNull.Value ? Convert.ToInt32(_sqlDataReader["Age"]) : 0;
                    response.getCitizenInformationByNIC.Profession = _sqlDataReader["Profession"] != DBNull.Value ? _sqlDataReader["Profession"].ToString() : string.Empty;
                    response.getCitizenInformationByNIC.Email = _sqlDataReader["Email"] != DBNull.Value ? _sqlDataReader["Email"].ToString() : string.Empty;
                    response.getCitizenInformationByNIC.Qualification = _sqlDataReader["Qualification"] != DBNull.Value ? _sqlDataReader["Qualification"].ToString() : string.Empty;
                }
                else
                {
                    response.Message = "No Citizen Found";
                }
            }
        }
    }
    catch (Exception ex)
    {
        response.IsSuccess = false;
        response.Message = "Exception Message : " + ex.Message;
    }
    finally
    {
        await _mySqlConnection.CloseAsync();
        await _mySqlConnection.DisposeAsync();
    }

    return response;
}
```

Figure 10 - Citizen Information by NIC method in the AuthDL.cs

Get List of Citizen Information By Qualification

- The below two screenshots represent the List of Citizen Information by their Qualification.

```
/// <summary>
/// Get Citizen Information By Qualification
/// </summary>
/// <param name="request"></param>
/// <returns></returns>
[AllowAnonymous]
[HttpGet]
[Route("Get/{Qualification}")]
0 references
public async Task<IActionResult> CitizenInformationByQualification([FromHeader] CitizenInformationByQualificationRequest request)
{
    CitizenInformationByQualificationResponse response = null;
    try
    {
        response = await _authDL.GetCitizenInformationByQualification(request);
    }
    catch (Exception ex)
    {
        response.IsSuccess = false;
        response.Message = "Exception Message : " + ex.Message;
    }

    return Ok(response);
}
```

Figure 11 - Citizen Information by Qualification method in the AuthController.cs

```
/// <summary>
/// Get Citizen Information By Qualification
/// </summary>
/// <param name="request"></param>
/// <returns></returns>
3 references
public async Task<CitizenInformationByQualificationResponse> GetCitizenInformationByQualification(CitizenInformationByQualificationRequest request)
{
    CitizenInformationByQualificationResponse response = new CitizenInformationByQualificationResponse();
    response.getCitizenInformationByQualification = new List<GetCitizenInformationByQualification>();

    response.IsSuccess = true;
    response.Message = "Successful";

    try
    {
        //SQL Query
        string SqlQuery = @"SELECT * FROM slbfe.user_details WHERE Qualification = @Qualification AND Affiliation = 'Citizen'";

        //Check Null Or Empty
        if (string.IsNullOrEmpty(request.Qualification))
        {
            response.Message = "Qualification cannot Null or Empty";
        }

        using (MySqlCommand sqlCommand = new MySqlCommand(SqlQuery, _mySqlConnection))
        {
            sqlCommand.CommandType = System.Data.CommandType.Text;
            sqlCommand.CommandTimeout = ConnectionTimeOut;
            sqlCommand.Parameters.AddWithValue("@Qualification", request.Qualification);
            await _mySqlConnection.OpenAsync();

            using (DbDataReader _sqlDataReader = await sqlCommand.ExecuteReaderAsync())
            {
                if (_sqlDataReader.HasRows)
                {
                    while (await _sqlDataReader.ReadAsync())
                    {
                        GetCitizenInformationByQualification getResponse = new GetCitizenInformationByQualification();

                        getResponse.NIC = _sqlDataReader["NIC"] != DBNull.Value ? _sqlDataReader["NIC"].ToString() : string.Empty;

                        getResponse.Name = _sqlDataReader["Name"] != DBNull.Value ? _sqlDataReader["Name"].ToString() : string.Empty;

                        getResponse.Address = _sqlDataReader["Address"] != DBNull.Value ? _sqlDataReader["Address"].ToString() : string.Empty;

                        getResponse.Age = _sqlDataReader["Age"] != DBNull.Value ? Convert.ToInt32(_sqlDataReader["Age"]) : 0;

                        getResponse.Profession = _sqlDataReader["Profession"] != DBNull.Value ? _sqlDataReader["Profession"].ToString() : string.Empty;

                        getResponse.Email = _sqlDataReader["Email"] != DBNull.Value ? _sqlDataReader["Email"].ToString() : string.Empty;

                        response.getCitizenInformationByQualification.Add(getResponse);
                    }
                }
                else
                {
                    response.Message = "No data Return";
                }
            }
        }
    }
    catch (Exception ex)
    {
        response.IsSuccess = false;
        response.Message = "Exception Message : " + ex.Message;
    }
    finally
    {
        await _mySqlConnection.CloseAsync();
        await _mySqlConnection.DisposeAsync();
    }

    return response;
}
```

Figure 12 - Citizen Information by Qualification method in the AuthDL.cs

Update Qualification By NIC

- The below two screenshots represent the Update Citizen Qualification by their NIC.

```
/// <summary>
/// Update Citizen's Qualification
/// </summary>
/// <param name="request"></param>
/// <returns></returns>
[AllowAnonymous]
[HttpPut]
[Route("UpdateQualification/{NIC}")]
0 references
public async Task<IActionResult> UpdateCitizenInformation(UpdateCitizenInformationRequest request)
{
    UpdateCitizenInformationResponse response = null;
    try
    {
        response = await _authDL.UpdateCitizenInformationRequest(request);
    }
    catch (Exception ex)
    {
        response.IsSuccess = false;
        response.Message = "Exception Message : " + ex.Message;
    }

    return Ok(response);
}
```

Figure 13 - Update Citizen Information method in the AuthController.cs

```
/// <summary>
/// Update Citizen Qualification
/// </summary>
/// <param name="request"></param>
/// <returns></returns>
3 references
public async Task<UpdateCitizenInformationResponse> UpdateCitizenInformationRequest(UpdateCitizenInformationRequest request)
{
    UpdateCitizenInformationResponse response = new UpdateCitizenInformationResponse();

    response.IsSuccess = true;
    response.Message = "Successful";

    try
    {
        if (_mySqlConnection != null)
        {
            //SQL Query
            string SqlQuery = @"UPDATE slbfe.user_details SET Qualification=@Qualification where NIC=@NIC ";

            using (MySqlCommand sqlCommand = new MySqlCommand(SqlQuery, _mySqlConnection))
            {
                sqlCommand.CommandType = System.Data.CommandType.Text;
                sqlCommand.CommandTimeout = ConnectionTimeOut;
                sqlCommand.Parameters.AddWithValue("@NIC", request.NIC);
                sqlCommand.Parameters.AddWithValue("@Qualification", request.Qualification);
                await _mySqlConnection.OpenAsync();
                int Status = await sqlCommand.ExecuteNonQueryAsync();

                if (Status <= 0)
                {
                    response.IsSuccess = false;
                    response.Message = "Informations Not Update";
                }
            }
        }

    }
    catch (Exception ex)
    {
        response.IsSuccess = false;
        response.Message = "Exception Message : " + ex.Message;
    }
    finally
    {
        await _mySqlConnection.CloseAsync();
        await _mySqlConnection.DisposeAsync();
    }

    return response;
}
```

Figure 14 - Update Citizen Information method in the AuthDL.cs

Upload Citizen Documents

- The below screenshot represents the Upload Citizen Documents.

```
/// <summary>
/// Upload Citizen's BirthCertificate, CV, Passport
/// </summary>
/// <param name="file"></param>
/// <returns></returns>
[AllowAnonymous]
[HttpPost]
0 references
public async Task<string> UploadCitizenDocuments(IFormFile file)
{
    string filename = "";

    try
    {
        var extension = "." + file.FileName.Split('.')[file.FileName.Split('.').Length - 1];
        filename = "File_" + DateTime.Now.TimeOfDay.Milliseconds + extension;
        var pathBuilt = Path.Combine(Directory.GetCurrentDirectory(), "Files");

        if (!Directory.Exists(pathBuilt))
        {
            Directory.CreateDirectory(pathBuilt);
        }

        var path = Path.Combine(Directory.GetCurrentDirectory(), "Files", filename);

        using (var stream = new FileStream(path, FileMode.Create))
        {
            await file.CopyToAsync(stream);
        }
    }
    catch (Exception ex)
    {
        return ex.Message.ToString();
    }

    return filename;
}
```

Figure 15 - Upload Citizen Documents method in the AuthController.cs

Delete Citizen Account by UserId

- The below two screenshots represent the Delete Citizen Account by UserId.

```
/// <summary>
/// Delete Citizen Account
/// </summary>
/// <param name="request"></param>
/// <returns></returns>
[AllowAnonymous]
[HttpDelete]
[Route("Delete/{UserId}")]
0 references
public async Task<IActionResult> DeleteCitizen([FromHeader] DeleteCitizenRequest request)
{
    DeleteCitizenResponse response = null;
    try
    {
        response = await _authDL.DeleteCitizen(request);
    }
    catch (Exception ex)
    {
        response.IsSuccess = false;
        response.Message = "Exception Message : " + ex.Message;
    }

    return Ok(response);
}
```

Figure 16 - Delete Citizen method in the AuthController.cs

```
/// <summary>
/// Delete Citizen Account
/// </summary>
/// <param name="request"></param>
/// <returns></returns>
3 references
public async Task<DeleteCitizenResponse> DeleteCitizen(DeleteCitizenRequest request)
{
    DeleteCitizenResponse response = new DeleteCitizenResponse();

    response.IsSuccess = true;
    response.Message = "Successful";

    try
    {
        if (_mySqlConnection != null)
        {
            //SQL Query
            string SqlQuery = @"DELETE FROM slbfe.user_details WHERE UserId = @UserId AND Affiliation = 'Citizen'";

            using (MySqlCommand sqlCommand = new MySqlCommand(SqlQuery, _mySqlConnection))
            {
                sqlCommand.CommandType = System.Data.CommandType.Text;
                sqlCommand.CommandTimeout = ConnectionTimeOut;
                sqlCommand.Parameters.AddWithValue("?UserId", request.UserId);
                await _mySqlConnection.OpenAsync();
                int Status = await sqlCommand.ExecuteNonQueryAsync();

                if (Status <= 0)
                {
                    response.IsSuccess = false;
                    response.Message = "Please check the provided information carefully";
                }
            }
        }
    }
    catch (Exception ex)
    {
    catch (Exception ex)
    {
        response.IsSuccess = false;
        response.Message = "Exception Message : " + ex.Message;
    }
    finally
    {
        await _mySqlConnection.CloseAsync();
        await _mySqlConnection.DisposeAsync();
    }

    return response;
}
```

Figure 17 - Delete Citizen method in the AuthDL.cs

Read Complaints

- The below two screenshots represent the List of Citizen's Complaints.

```
/// <summary>
/// Get Complaints List
/// </summary>
/// <returns></returns>
[AllowAnonymous]
[HttpGet]
0 references
public async Task<IActionResult> ReadComplaints()
{
    ReadComplaintsResponse response = null;
    try
    {
        response = await _authDL.ReadComplaints();
    }
    catch (Exception ex)
    {
        response.IsSuccess = false;
        response.Message = "Exception Message : " + ex.Message;
    }

    return Ok(response);
}
```

Figure 18 - Read Complaints method in the AuthController.cs

```
/// <summary>
/// Get Complaints List
/// </summary>
/// <returns></returns>
3 references
public async Task<ReadComplaintsResponse> ReadComplaints()
{
    ReadComplaintsResponse response = new ReadComplaintsResponse();
    response.readComplaints = new List<ReadComplaints>();

    response.IsSuccess = true;
    response.Message = "Successful";

    try
    {
        //SQL Query
        string SqlQuery = @"SELECT * FROM slbfe.complaints ";

        using (MySqlCommand sqlCommand = new MySqlCommand(SqlQuery, _mySqlConnection))
        {
            await _mySqlConnection.OpenAsync();

            using (DbDataReader _sqlDataReader = await sqlCommand.ExecuteReaderAsync())
            {
                if (_sqlDataReader.HasRows)
                {
                    while (await _sqlDataReader.ReadAsync())
                    {
                        ReadComplaints getResponse = new ReadComplaints();

                        getResponse.ComplaintId = _sqlDataReader["ComplaintId"] != DBNull.Value ? Convert.ToInt32(_sqlDataReader["ComplaintId"]) : 0;

                        getResponse.Complaint = _sqlDataReader["Complaint"] != DBNull.Value ? _sqlDataReader["Complaint"].ToString() : string.Empty;

                        getResponse.Reply = _sqlDataReader["Reply"] != DBNull.Value ? _sqlDataReader["Reply"].ToString() : string.Empty;

                        response.readComplaints.Add(getResponse);
                    }
                }
                }
                else
                {
                    response.Message = "No data Return";
                }
            }
        }
    }
    catch (Exception ex)
    {
        response.IsSuccess = false;
        response.Message = "Exception Message : " + ex.Message;
    }
    finally
    {
        await _mySqlConnection.CloseAsync();
        await _mySqlConnection.DisposeAsync();
    }

    return response;
}
```

Figure 19 - Read Complaints method in the AuthDL.cs

Create a Complaint

- The below two screenshots represent the create a Complaint.

```
/// <summary>
/// Ceate a Complaint
/// </summary>
/// <param name="request"></param>
/// <returns></returns>
[AllowAnonymous]
[HttpPost]
0 references
public async Task<ActionResult> CreateComplaint([FromBody] CreateComplaintRequest request)
{
    CreateComplaintResponse response = new CreateComplaintResponse();
    try
    {
        response = await _authDL.CreateComplaintInformation(request);
    }
    catch (Exception ex)
    {
        response.IsSuccess = false;
        response.Message = "Exception Message : " + ex.Message;
    }

    return Ok(response);
}
```

Figure 20 - Create Complaint method in the AuthController.cs

```
/// <summary>
/// Create Complaint
/// </summary>
/// <param name="request"></param>
/// <returns></returns>
3 references
public async Task<CreateComplaintResponse> CreateComplaintInformation(CreateComplaintRequest request)
{
    CreateComplaintResponse response = new CreateComplaintResponse();

    response.IsSuccess = true;
    response.Message = "Successful";

    try
    {
        if (_mySqlConnection.State != System.Data.ConnectionState.Open)
        {
            await _mySqlConnection.OpenAsync();
        }

        //SQL Query
        string SqlQuery = @"INSERT INTO slbfe.complaints (Complaint) VALUES (@Complaint)";

        using (MySqlCommand sqlCommand = new MySqlCommand(SqlQuery, _mySqlConnection))
        {
            sqlCommand.CommandType = System.Data.CommandType.Text;
            sqlCommand.CommandTimeout = 180;
            sqlCommand.Parameters.AddWithValue("@Complaint", request.Complaint);
            int Status = await sqlCommand.ExecuteNonQueryAsync();

            if (Status <= 0)
            {
                response.IsSuccess = false;
                response.Message = "Something Went Wrong";
                return response;
            }
        }
    }
    catch (Exception ex)
    {
        response.IsSuccess = false;
        response.Message = ex.Message;
    }
    finally
    {
        await _mySqlConnection.CloseAsync();
        await _mySqlConnection.DisposeAsync();
    }

    return response;
}
```

Figure 21 - Create Complaint method in the AuthDL.cs

Reply for a Complaint

- The below two screenshots represent the Reply to the Citizen's Complaints.

```
/// <summary>
/// Reply for Complaints
/// </summary>
/// <param name="request"></param>
/// <returns></returns>
[AllowAnonymous]
[HttpPut]
[Route("Update/{ComplaintId}")]
0 references
public async Task<IActionResult> UpdateComplaintInformation(UpdateComplaintInformationRequest request)
{
    UpdateComplaintInformationResponse response = null;
    try
    {
        response = await _authDL.UpdateComplaintInformationRequest(request);
    }
    catch (Exception ex)
    {
        response.IsSuccess = false;
        response.Message = "Exception Message : " + ex.Message;
    }

    return Ok(response);
}
```

Figure 22 - Update Complaint Information method in the AuthController.cs

```
/// <summary>
/// Reply for Complaint
/// </summary>
/// <param name="request"></param>
/// <returns></returns>
3 references
public async Task<UpdateComplaintInformationResponse> UpdateComplaintInformationRequest(UpdateComplaintInformationRequest request)
{
    UpdateComplaintInformationResponse response = new UpdateComplaintInformationResponse();

    response.IsSuccess = true;
    response.Message = "Successful";

    try
    {
        if (_mySqlConnection != null)
        {
            //SQL Query
            string SqlQuery = @"UPDATE slbfe.complaints SET Reply=@Reply where ComplaintId=@ComplaintId ";

            using (MySqlCommand sqlCommand = new MySqlCommand(SqlQuery, _mySqlConnection))
            {
                sqlCommand.CommandType = System.Data.CommandType.Text;
                sqlCommand.CommandTimeout = ConnectionTimeOut;
                sqlCommand.Parameters.AddWithValue("@ComplaintId", request.ComplaintId);
                sqlCommand.Parameters.AddWithValue("@Reply", request.Reply);
                await _mySqlConnection.OpenAsync();
                int Status = await sqlCommand.ExecuteNonQueryAsync();

                if (Status <= 0)
                {
                    response.IsSuccess = false;
                    response.Message = "Information Not Update";
                }
            }
        }
    }
    catch (Exception ex)
    {
        response.IsSuccess = false;
        response.Message = "Exception Message : " + ex.Message;
    }
    finally
    {
        await _mySqlConnection.CloseAsync();
        await _mySqlConnection.DisposeAsync();
    }

    return response;
}
```

Figure 23 - Update Complaint Information method in the AuthDL.cs

2.2.1.2. Mobile Application

- When creating the mobile application the same .NET backend was used and then the application and the APIs was called through the application.

Import Retrofit Dependency

- Imported Retrofit dependencies to make a request and response communication between the front end and the APIs.

Models

User model

- Defined the variables of the user using a public class.

```
package com.example.slbfe.model;

public class user {
    private final int UserId;
    private final String NIC;
    private final String Name;
    private final String Address;
    private final int Age;
    private final String Profession;
    private final String Email;
    private final String Password;
    private final String Affiliation;
    private final Byte BCertificate;
    private final Byte CV;
    private final Byte Passport;
    private final String Qualification;
```

- Assigned values for the defined variables.

```
    public user (int id, String nic, String name, String address, int age,
String profession, String email, String password, String affiliation, Byte
bCertificate, Byte cv, Byte passport, String qualification) {

        UserId = id;
        this.NIC = nic;
        this.Name = name;
        this.Address = address;
        Age = age;
        this.Profession = profession;
        this.Email = email;
        this.Password = password;
        this.Affiliation = affiliation;
        this.BCertificate = bCertificate;
        this.CV = cv;
        this.Passport = passport;
        this.Qualification = qualification;
    }
```

- Accessed the private variables through a public function.

```
    public int getUserId() {
        return id;
    }
    public String getNIC() {
        return nic;
    }
    public String getName() {
        return name;
    }
    public String getAddress() {
        return address;
    }
    public int getAge() {
        return age;
    }
    public String getProfession() {
        return profession;
    }
    public String getEmail() {
        return email;
    }
    public String getPassword() {
        return password;
    }
    public String getAffiliation() {
        return affiliation;
    }
    public String getQualification() {
        return qualification;
    }
}
```

Complaint Model

```
package com.example.slbfe.model;

public class complaint {
    private final int ComplaintId;
    private final String Complaint;
    private final String Reply;

    public complaint(int id, String complaint, String reply) {

        ComplaintId = id;
        this.Complaint = complaint;
        this.Reply = reply;
    }

    public int getComplaintId() {
        return id;
    }

    public String getComplaint() {
        return complaint;
    }

    public String getReply() {
        return reply;
    }
}
```

Admin Model

```
package com.example.slbfe.model;

public class admin {
    private final int AdminId;
    private final String UserName;
    private final String Password;

    public admin(int id, String name, String password) {
        AdminId = id;
        this.UserName = name;
        this.Password = password;
    }

    public int getAdminId() {
        return id;
    }

    public String getUserName() {
        return name;
    }

    public String getPassword() {
        return password;
    }
}
```

Retrofit 2 service and Interface

- Calling APIs.

```
package com.example.slbfe.model;
import com.example.slbfe.model.user;
import com.example.slbfe.model.complaint;
import com.example.slbfe.model.admin;

import java.util.List;

import retrofit2.Call;
import retrofit2.http.GET;
import retrofit2.http.Path;



public interface IUserApi {
    @GET("Complaints")
    Call<List<complaint>> getComplaint();

    @GET("complaints/{id}/model")
    Call<List<model>> getModelByComplaint(@Path("id") int id);

    @GET("model")
    Call<List<model>> getModel();
}
```

Retrofit instance

- Connection setup

```
package com.example.slbfe.model;

import retrofit2.Retrofit;
import retrofit2.converter.gson.GsonConverterFactory;

public class RetrofitService {
    public static IUserApi Create() {
        Retrofit retrofit = new Retrofit.Builder()
                .baseUrl("http://10.0.2.2:5000/")
                .addConverterFactory(GsonConverterFactory.create())
                .build();
        return retrofit.create(IUserApi.class);
    }
}
```

API Manager

```
package com.example.slbfe.model;

import android.app.Application;
import com.example.slbfe.model.UserApiManager;

public class MainApplication extends Application {
    public static UserApiManager userApiManager;

    @Override
    public void onCreate(){
        super.onCreate();
        userApiManager = UserApiManager.getInstance();

}
}
```

ModelsAPI Manager

```
package com.example.slbfe.model.remote;
import android.graphics.ColorSpace;
import com.example.slbfe.model.IUserApi;
import com.example.slbfe.model.user;
import com.example.slbfe.model.complaint;
import com.example.slbfe.model.admin;


import java.util.List;

import retrofit2.Call;
import retrofit2.Callback;

public class ModelApiManager {

    private static IUserApi service;
    private static ModelApiManager apiManager;

    private ModelApiManager() {
        service = RetrofitService.Create();
    }
```

```
    public static ModelApiManager getInstance() {
        if (apiManager == null) {
            apiManager = new ModelApiManager();
        }
        return apiManager;
    }
    public void getUser(Callback<List<user>> callback){
        Call<List<user>> categoriesCall = service.getUser();
        categoriesCall.enqueue(callback);
    }
    public void getModelByUser(int id, Callback<List<Model>> callback){
        Call<List<Model>> ModelByUserCall = service.getByUser(id);
        ModelByUserCall.enqueue(callback);
    }
    public void getComplaint(Callback<List<Model>> callback){
        Call<List<Model>> ModelCall = service.getModel();
        ModelsCall.enqueue(callback);
    }
}
```

The repository

```
package com.example.slbfe.model.repository;
import android.graphics.ColorSpace;
import androidx.lifecycle.MutableLiveData;
import com.example.slbfe.model.user;
import com.example.slbfe.model.IUserApi;
import com.example.slbfe.model.complaint;
import com.example.slbfe.model.admin;

import java.util.List;

import retrofit2.Call;
import retrofit2.Callback;
import retrofit2.Response;

public class ModelRepository {

    private static volatile ModelRepository instance;

    private final ModelApiManager modelApiManager;

    private final MutableLiveData<List<user>> user = new MutableLiveData<>();
    private final MutableLiveData<List<complaint>> complaint =
    new MutableLiveData<>();
    private final MutableLiveData<List<admin>> admin =
    new MutableLiveData<>();

    private ModelRepository(ModelApiManager modelApiManager) {
        this.modelApiManager = modelApiManager;
    }
```

```
    public static ModelRepository getInstance(ModelApiManagermodelApiManager)
{
        if (instance == null) {
            instance = new ModelRepository(modelApiManager);
        }
        return instance;
    }

    public MutableLiveData<List<user>> getUser(){
        modelApiManager.getUser(new Callback<List<user>>() {
            @Override
            public void onResponse(Call<List<user>> call,
                                   Response<List<user>> response) {
                if (response.isSuccessful()){
                    List<user> body = response.body();
                    users.setValue(body);
                } else{
                    users.postValue(null);
                }
            }
```

```
            @Override
            public void onFailure(Call<List<user>> call, Throwable t) {
                user.postValue(null);
            }

        });

        return user;
    }

    public MutableLiveData<List<complaint>> getModelByComplaint(int id){
        modelApiManager.getModelByComplaint(id,
         new Callback<List<complaint>>() {
            @Override
            public void onResponse(Call<List<complaint>> call,
              Response<List<complaint>> response) {
                if (response.isSuccessful()){
                    List<complaint> body = response.body();
                    complaintById.setValue(body);
                } else{
                    complaintById.postValue(null);
                }
            }
            @Override
            public void onFailure(Call<List<complaint>> call, Throwable t) {
                complaintByCategory.postValue(null);
            }
        });

        return complaintById;
    }
```

```
    public MutableLiveData<List<admin>> getModel(){
        adminApiManager.getBlogs(new Callback<List<admin>>() {
            @Override
            public void onResponse(Call<List<admin>> call,
             Response<List<admin>> response) {
                if (response.isSuccessful()){
                    List<admin> body = response.body();
                    admin.setValue(body);
                } else{
                    admin.postValue(null);
                }
            }
            @Override
            public void onFailure(Call<List<admin>> call, Throwable t) {
                admin.postValue(null);
            }
        });

        return admin;
    }
}
```

2.2.2. Run-Time Screenshots

- The JSON output is displayed in the Swagger and the XML output is displayed with Postman.
- The integrated application is given to the users in which all the APIs are able to be controlled from the frontend.

2.2.2.1. Web APPLICATION

Citizen & Officer - Sign-Up Page

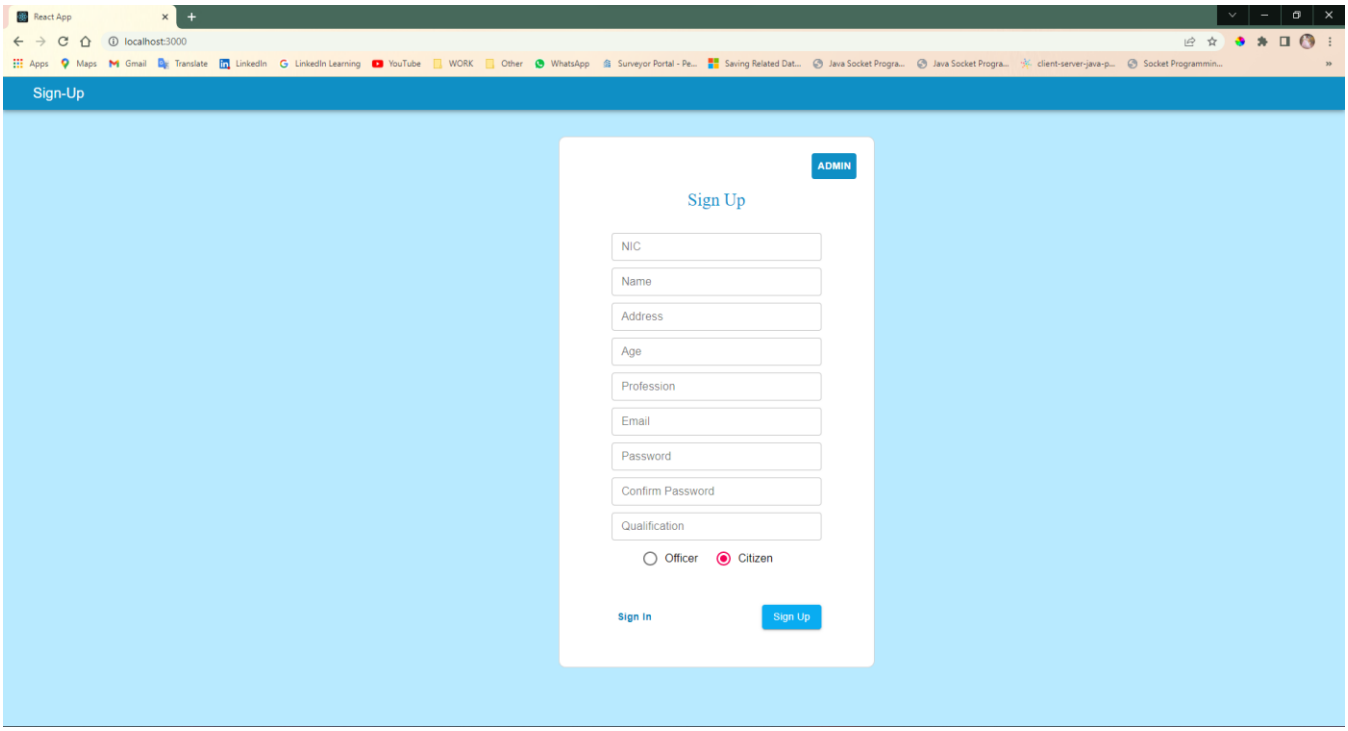


Figure 24 - Run-Time: Citizen & Officer Sign-Up User-Interface

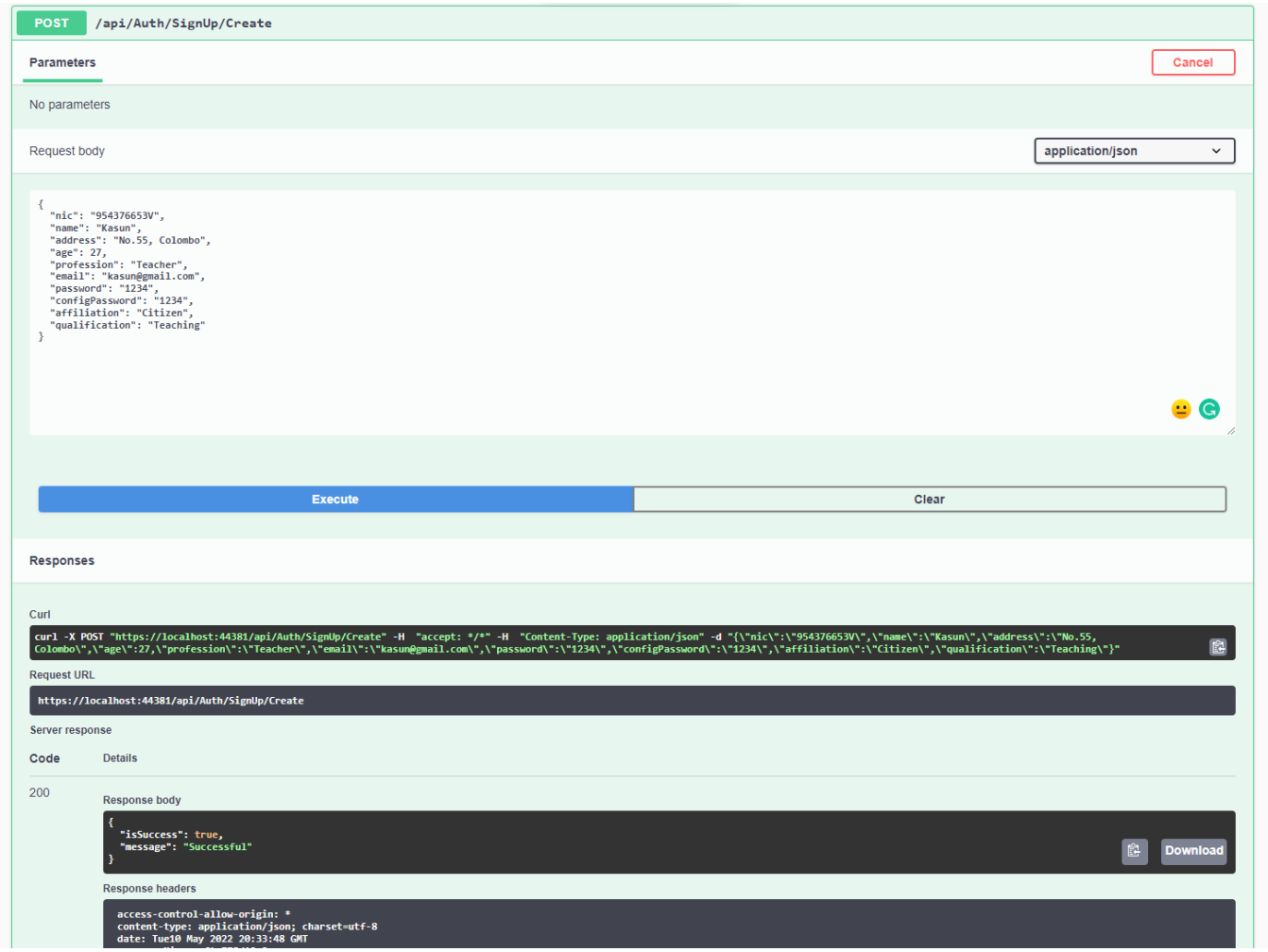


Figure 25 - Run-Time: Citizen & Officer Sign-Up with JSON result

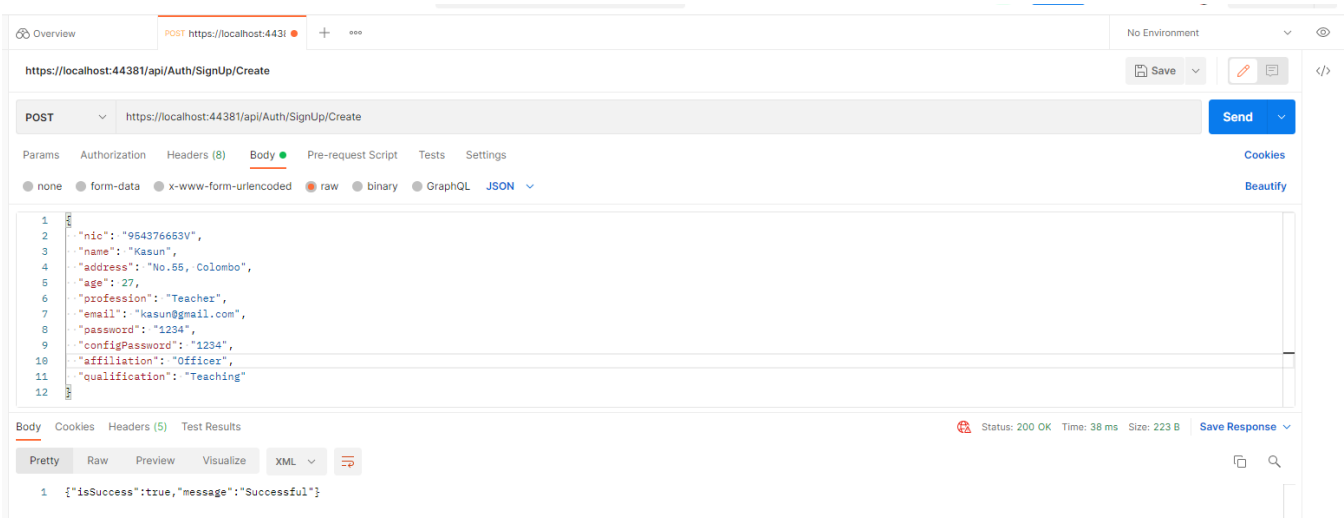


Figure 26 - Run-Time: Citizen & Officer Sign-Up with XML result

Citizen & Officer - Sign-In Page

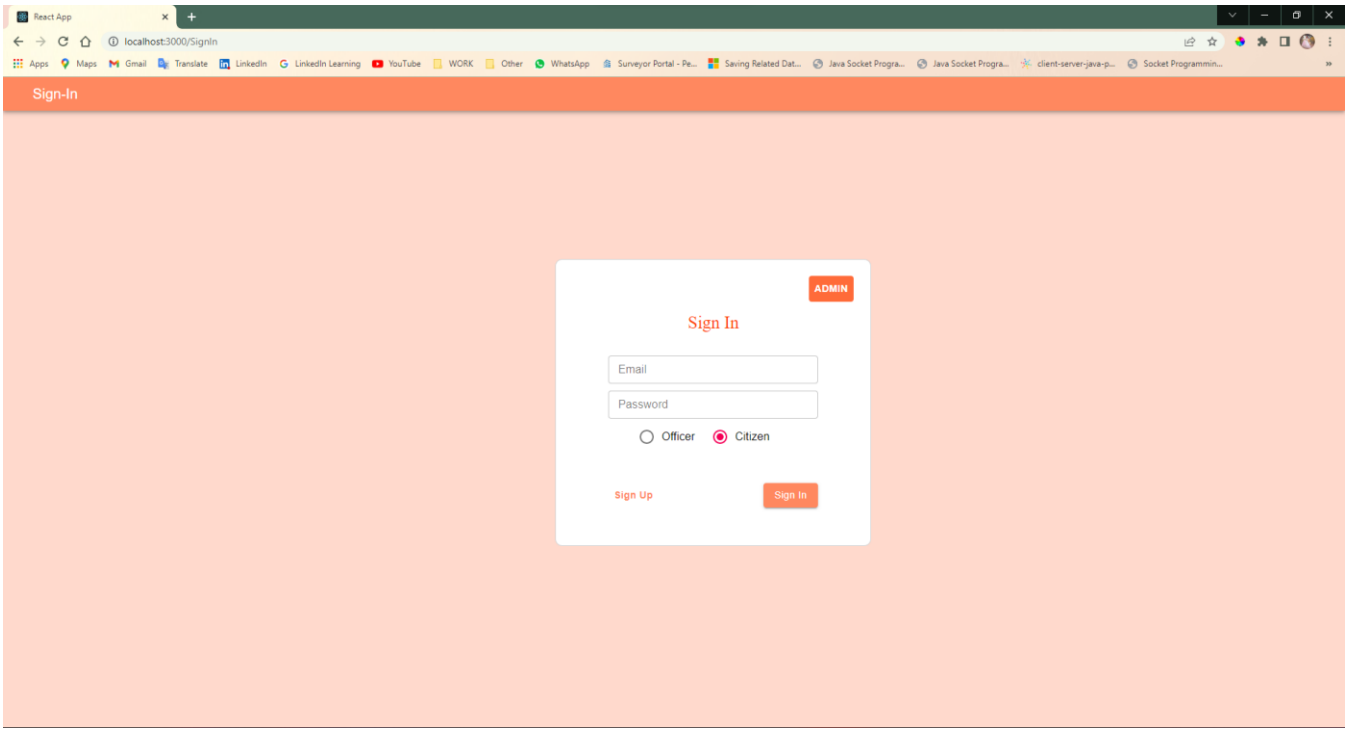


Figure 27 - Run-Time: Citizen & Officer Sign-In Use-Interface

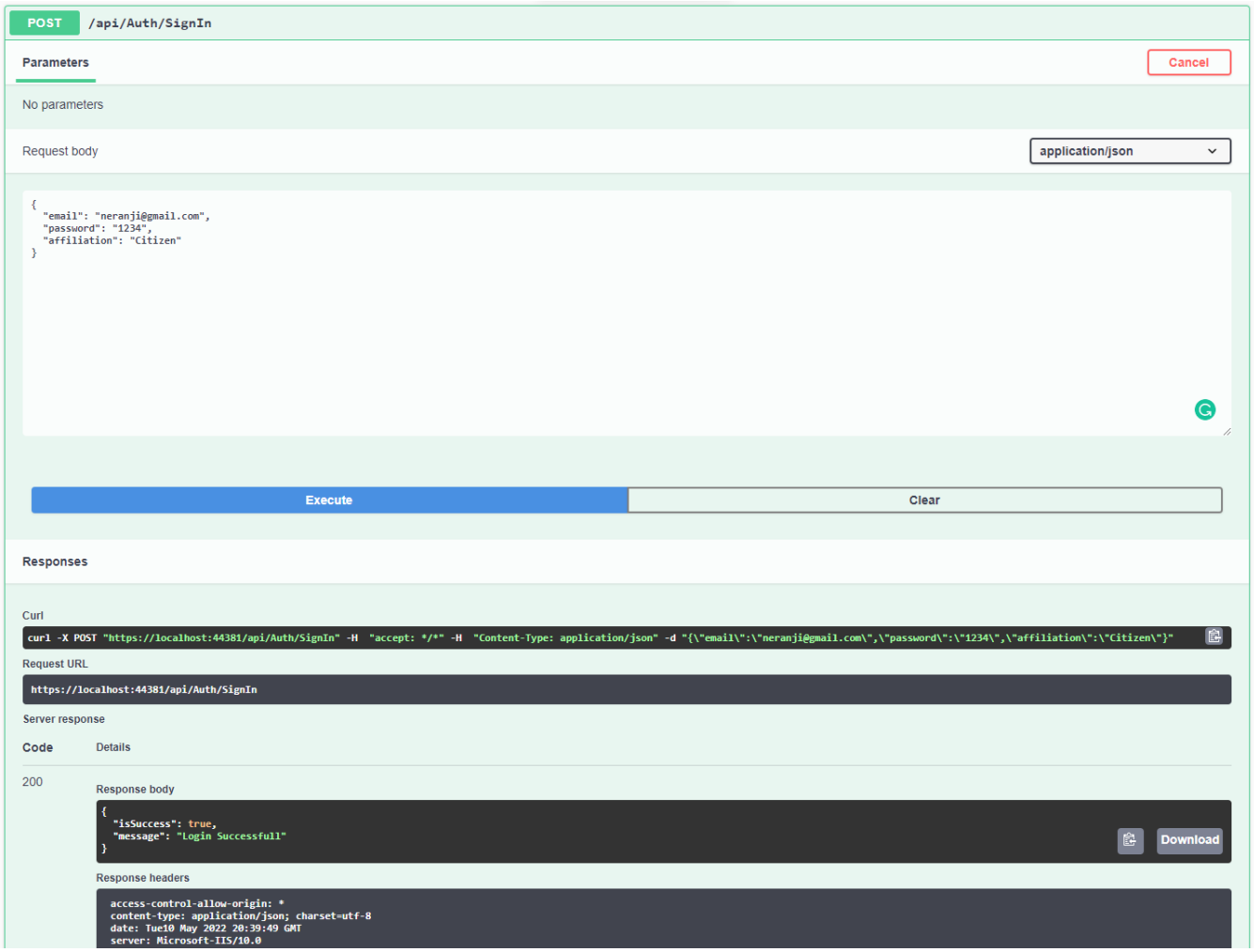


Figure 28 - Run-Time: Citizen & Officer Sign-In with JSON result

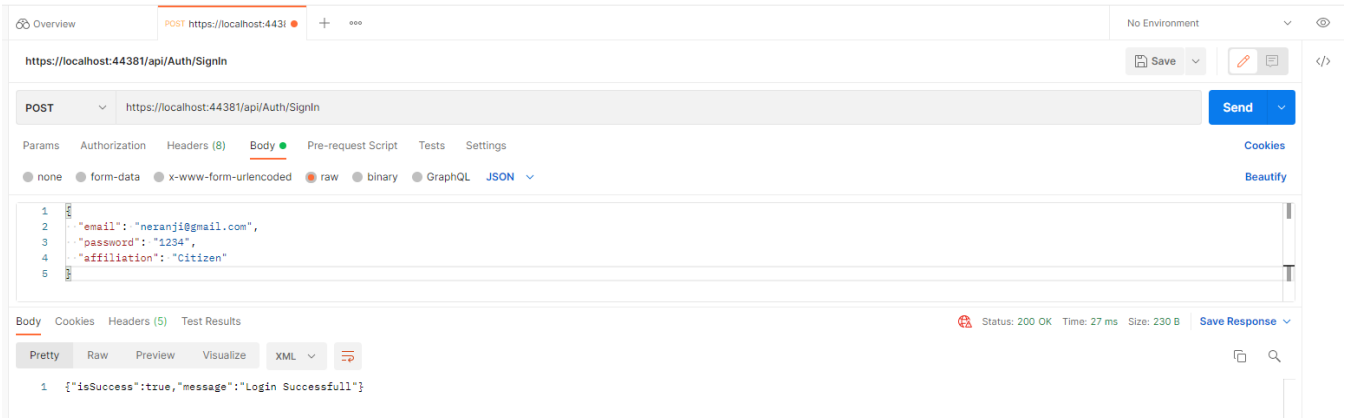


Figure 29 - Run-Time: Citizen & Officer Sign-In with XML result

Admin Sign-In Page

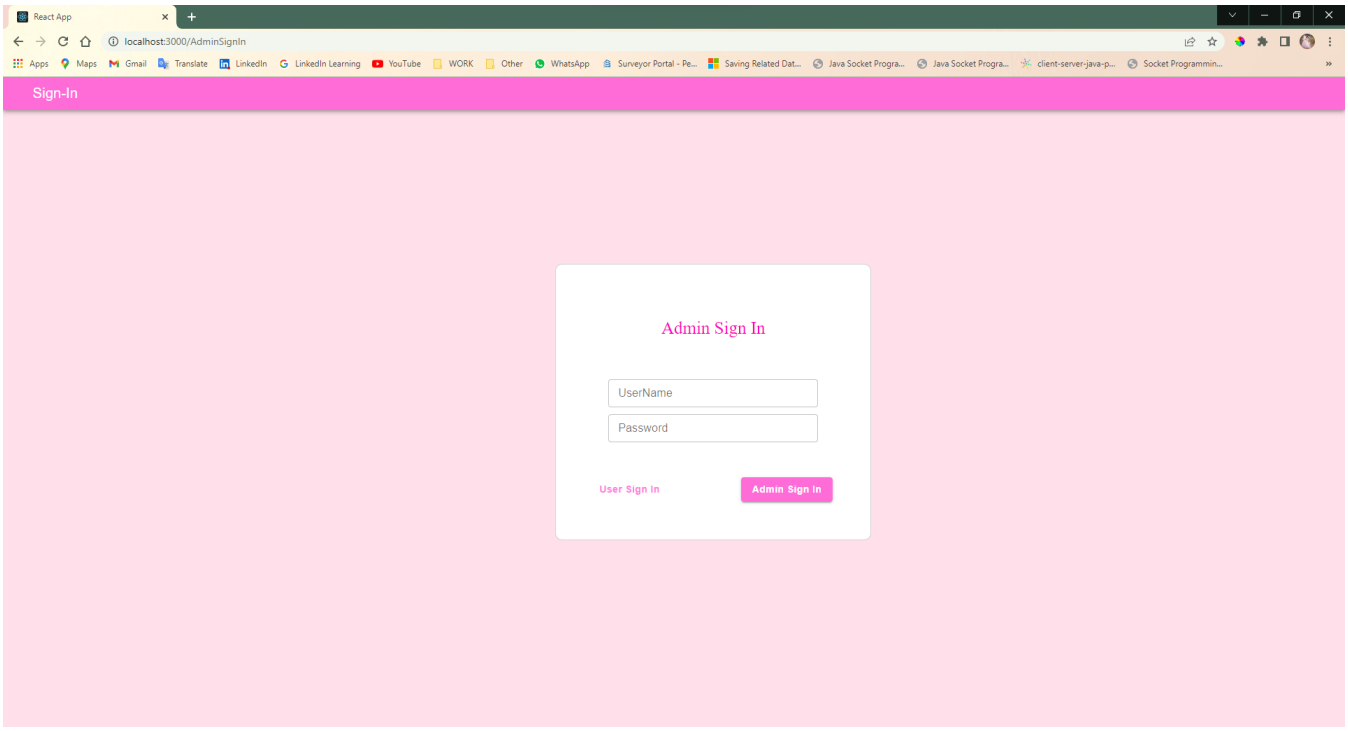


Figure 30 - Run-Time: Admin Sign-In User-Interface

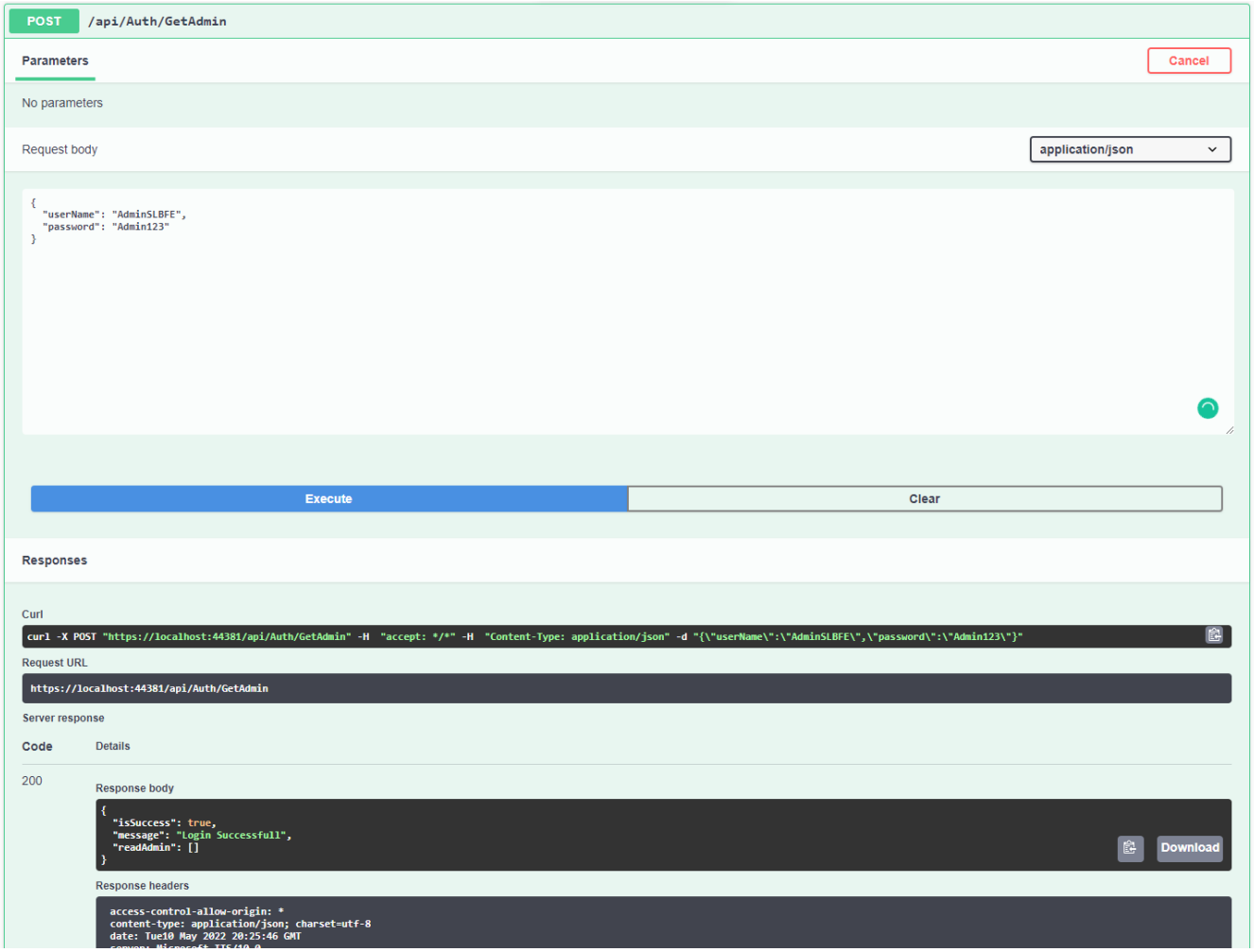


Figure 31 - Run-Time: Admin Sign-In with JSON result

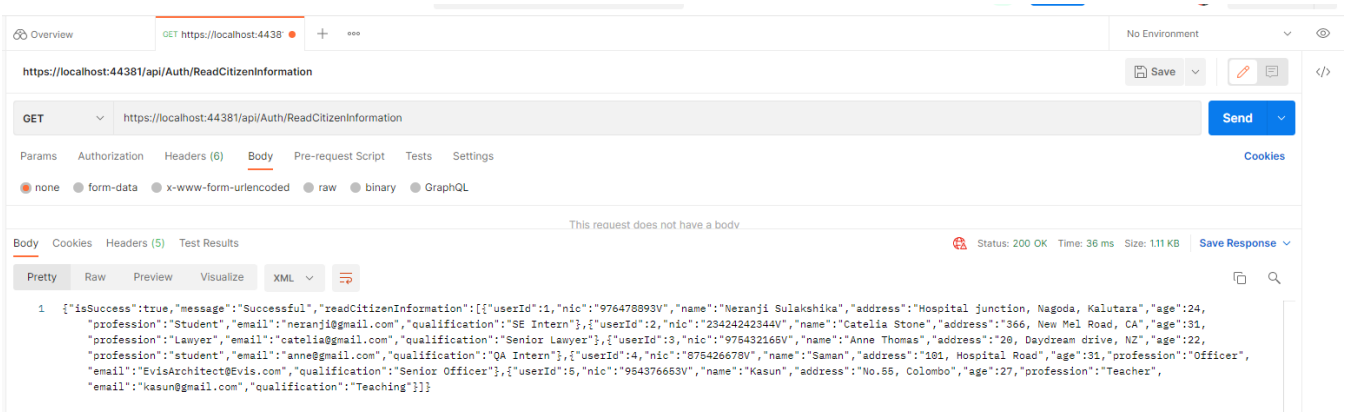


Figure 32 - Run-Time: Admin Sign-In with XML result

Admin Home Page

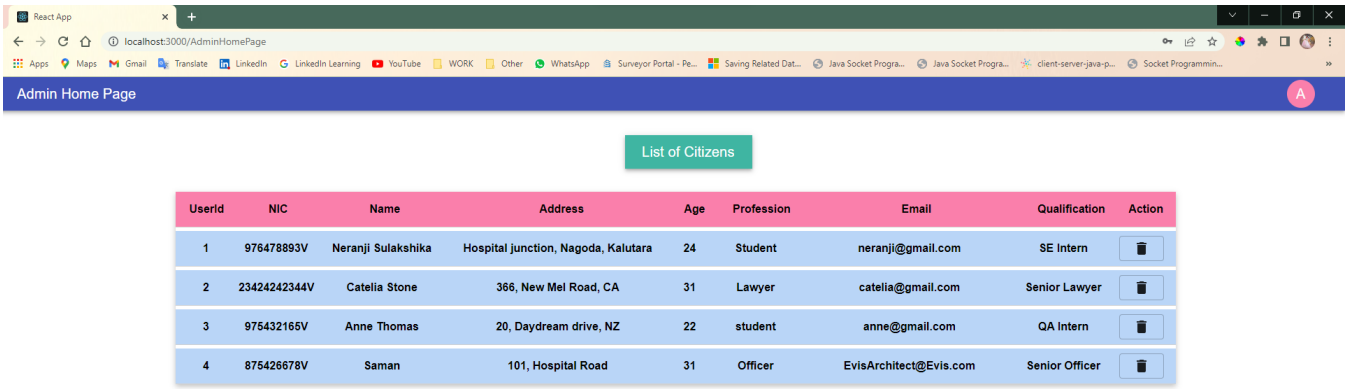


Figure 33 - Admin Home User-Interface

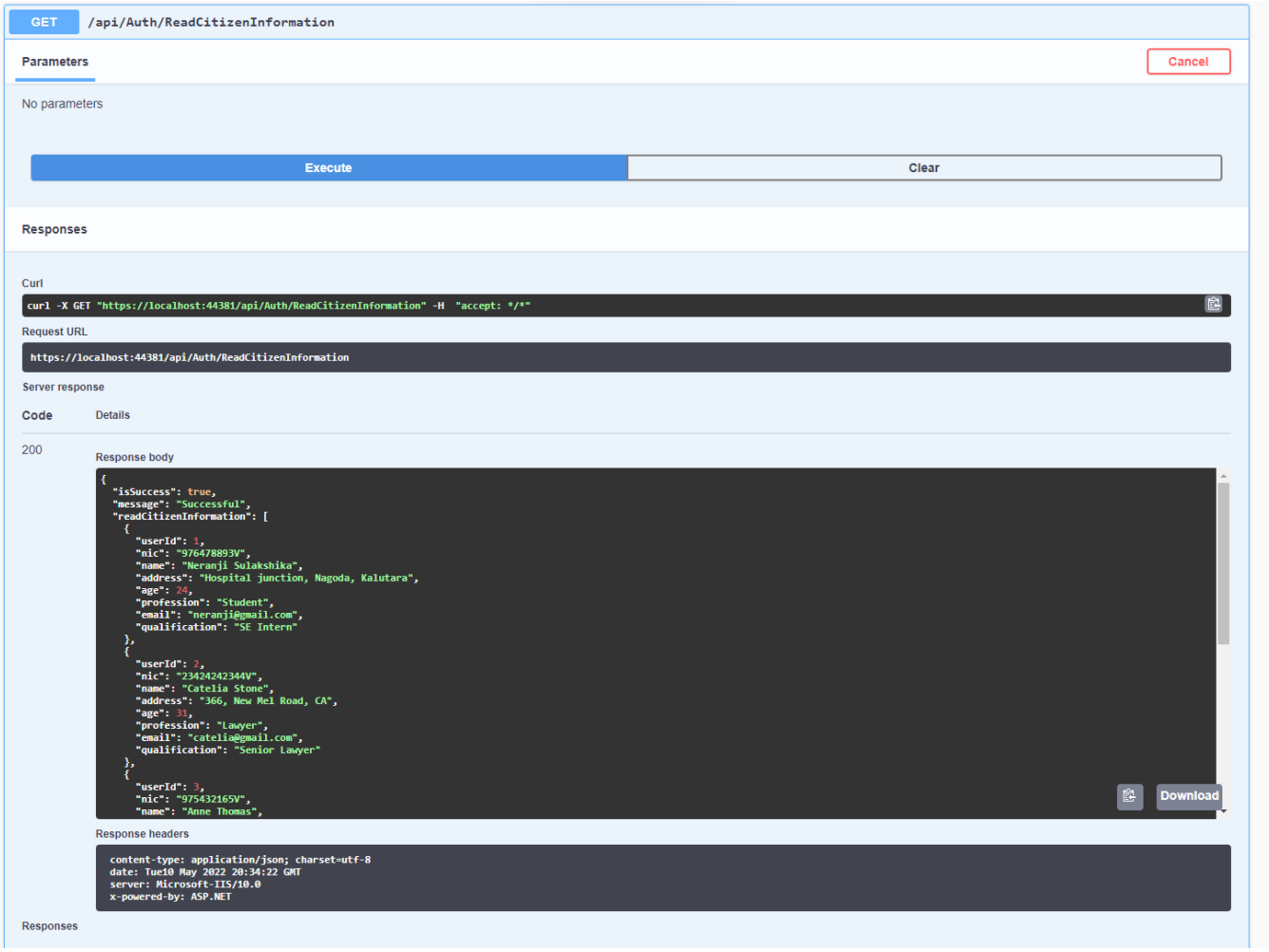


Figure 34 - Run-Time: List of Citizen Information with JSON result

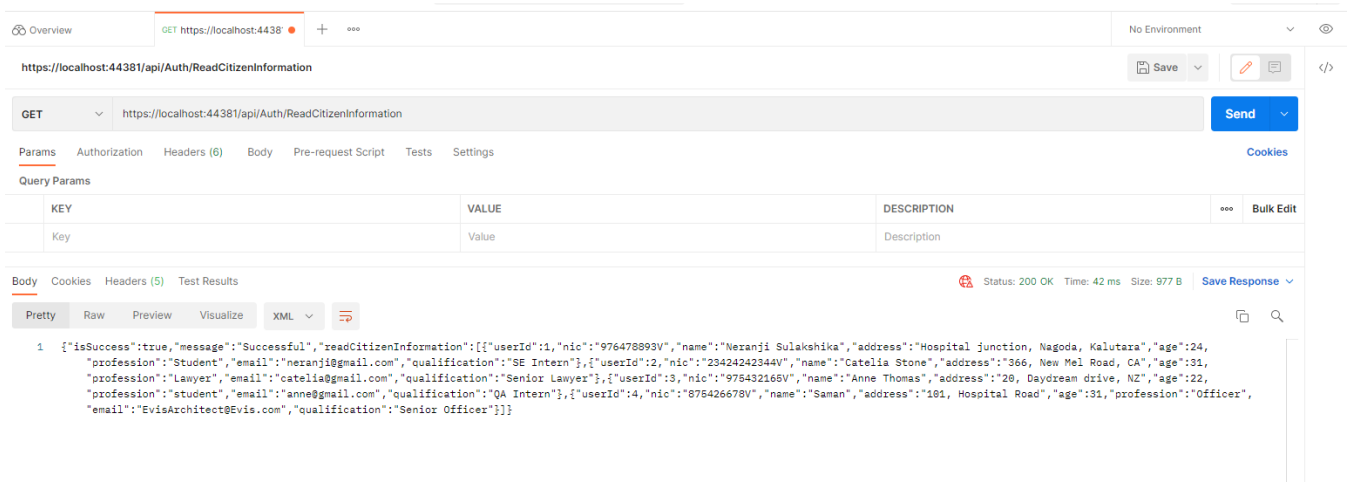


Figure 35 - Run-Time: List of Citizen Information with XML result

Delete Citizen Account

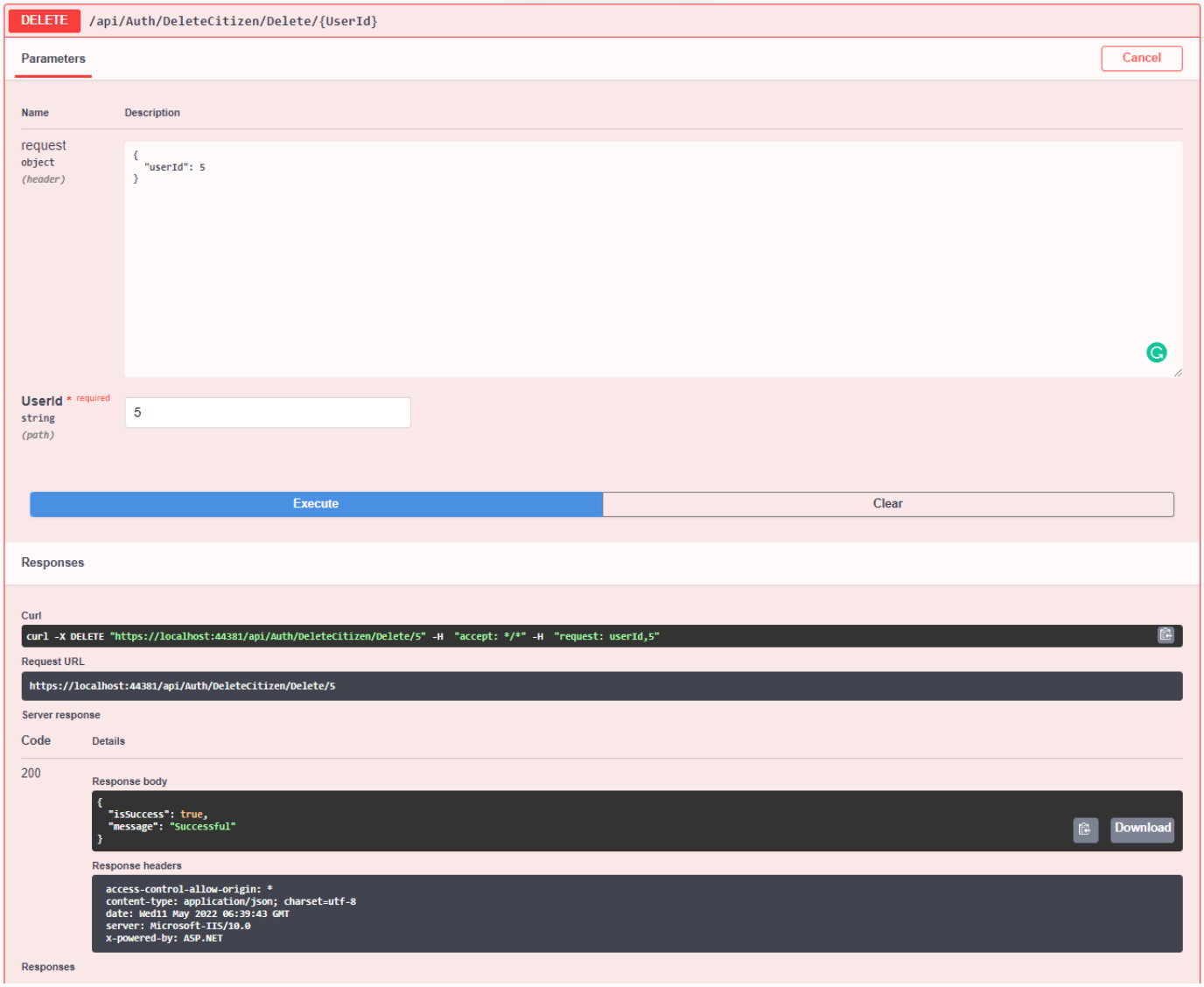


Figure 36 - Run-Time: Delete Citizen Account with JSON result

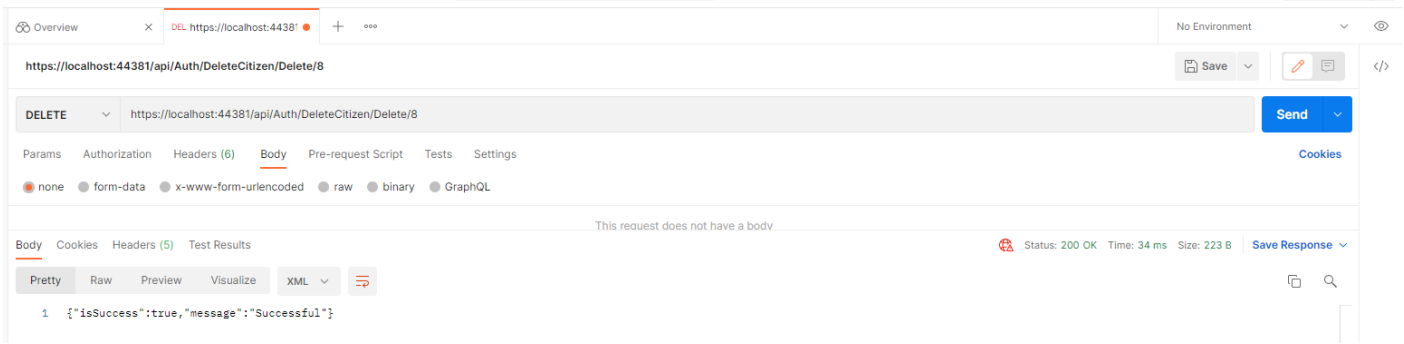


Figure 37 - Run-Time: Delete Citizen Account with XML result

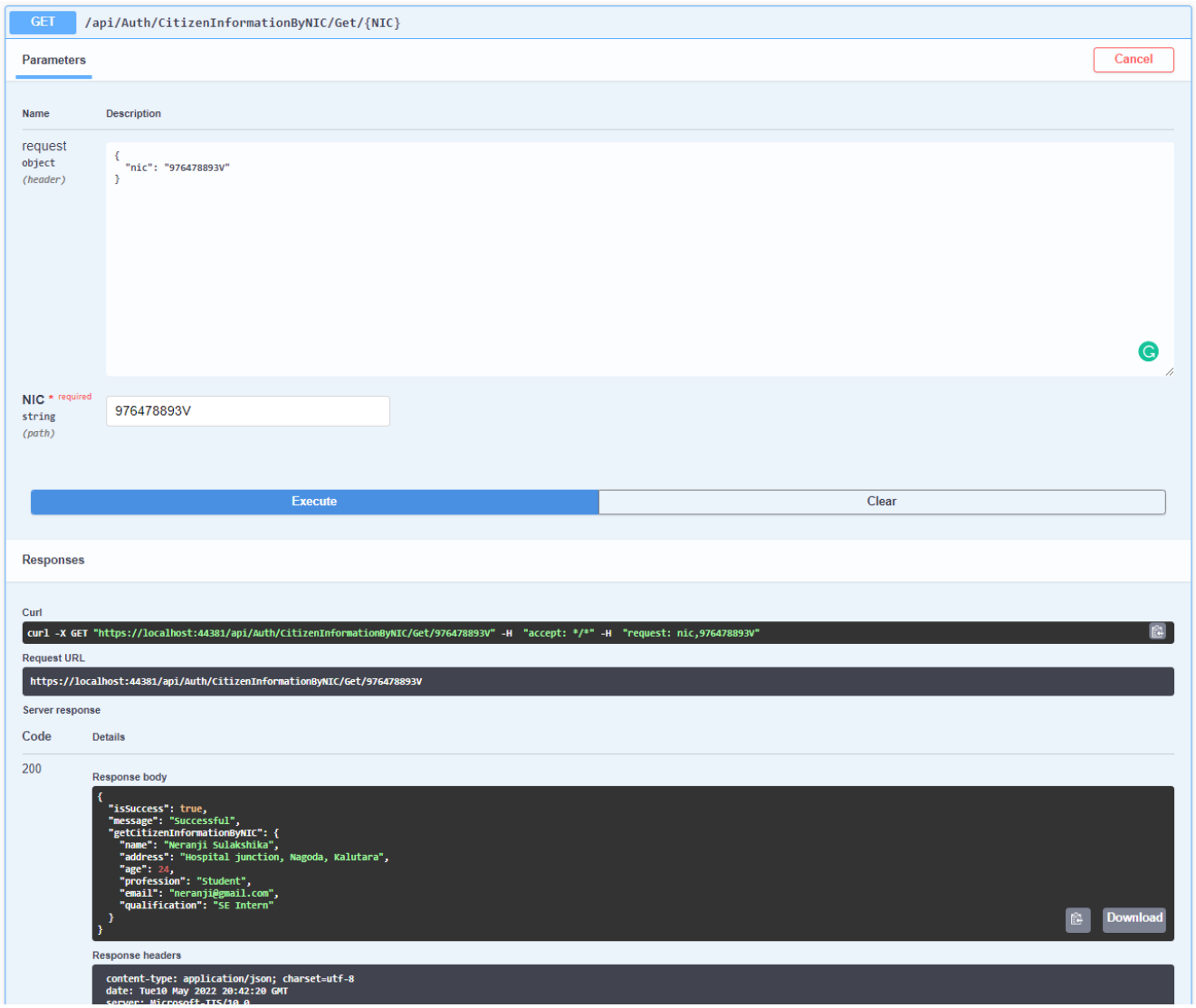


Figure 38 - Run-Time: Get Citizen Information by NIC with JSON result

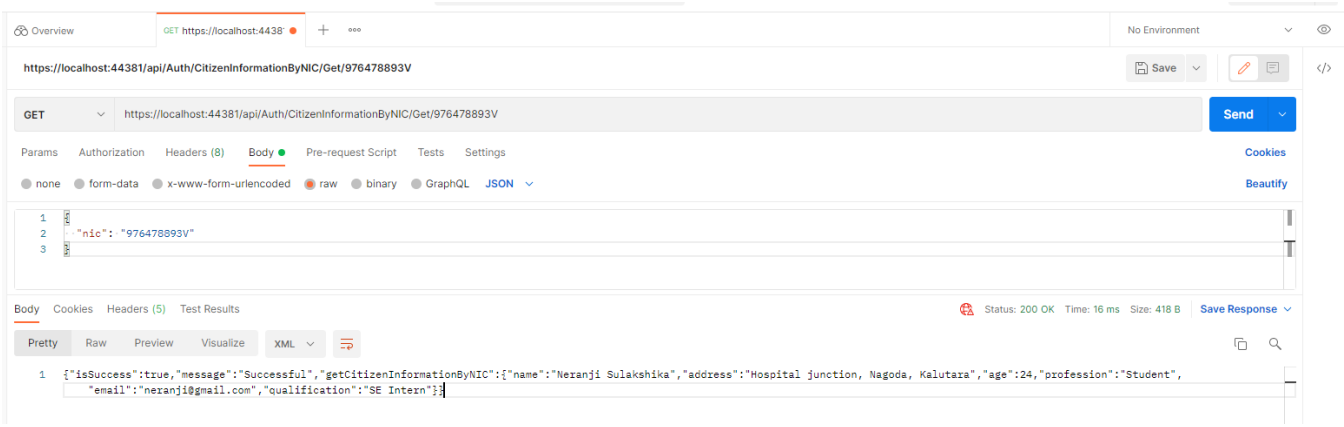


Figure 39 - Run-Time: Get Citizen Information by NIC with XML result

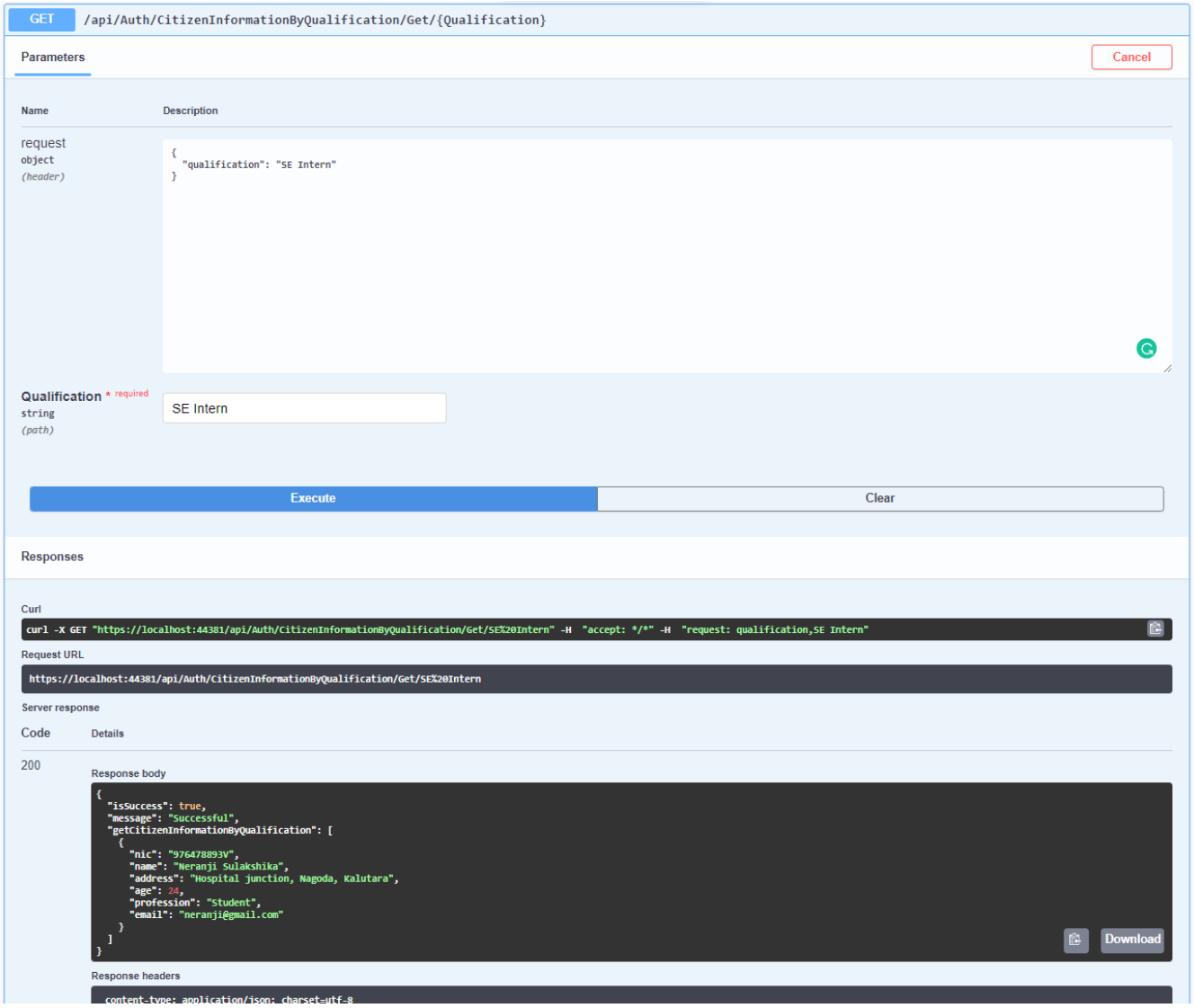


Figure 40 - Run-Time: Get Citizen Information by Qualification with JSON result

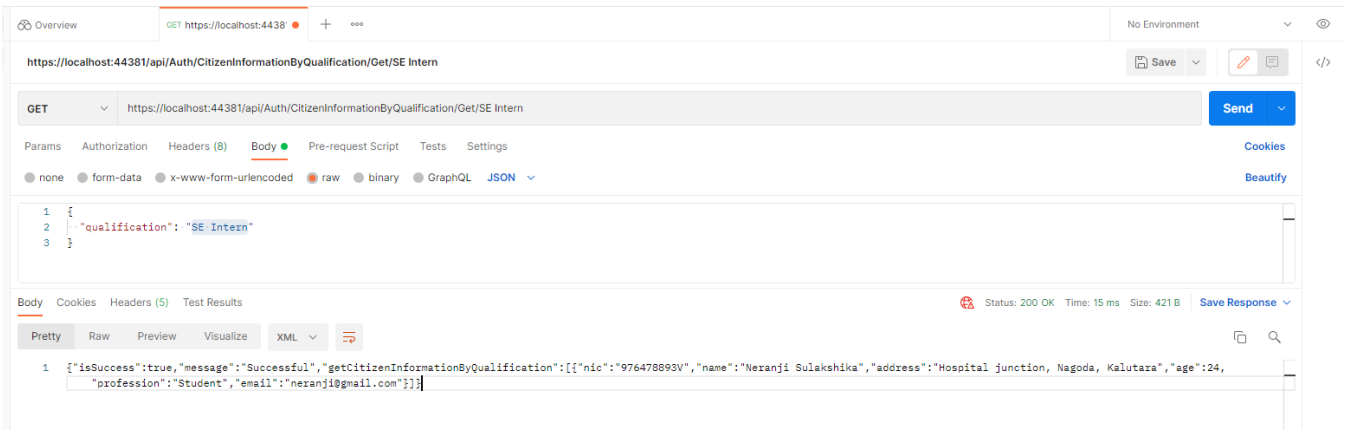


Figure 41 - Run-Time: Get Citizen Information by Qualification with XML result

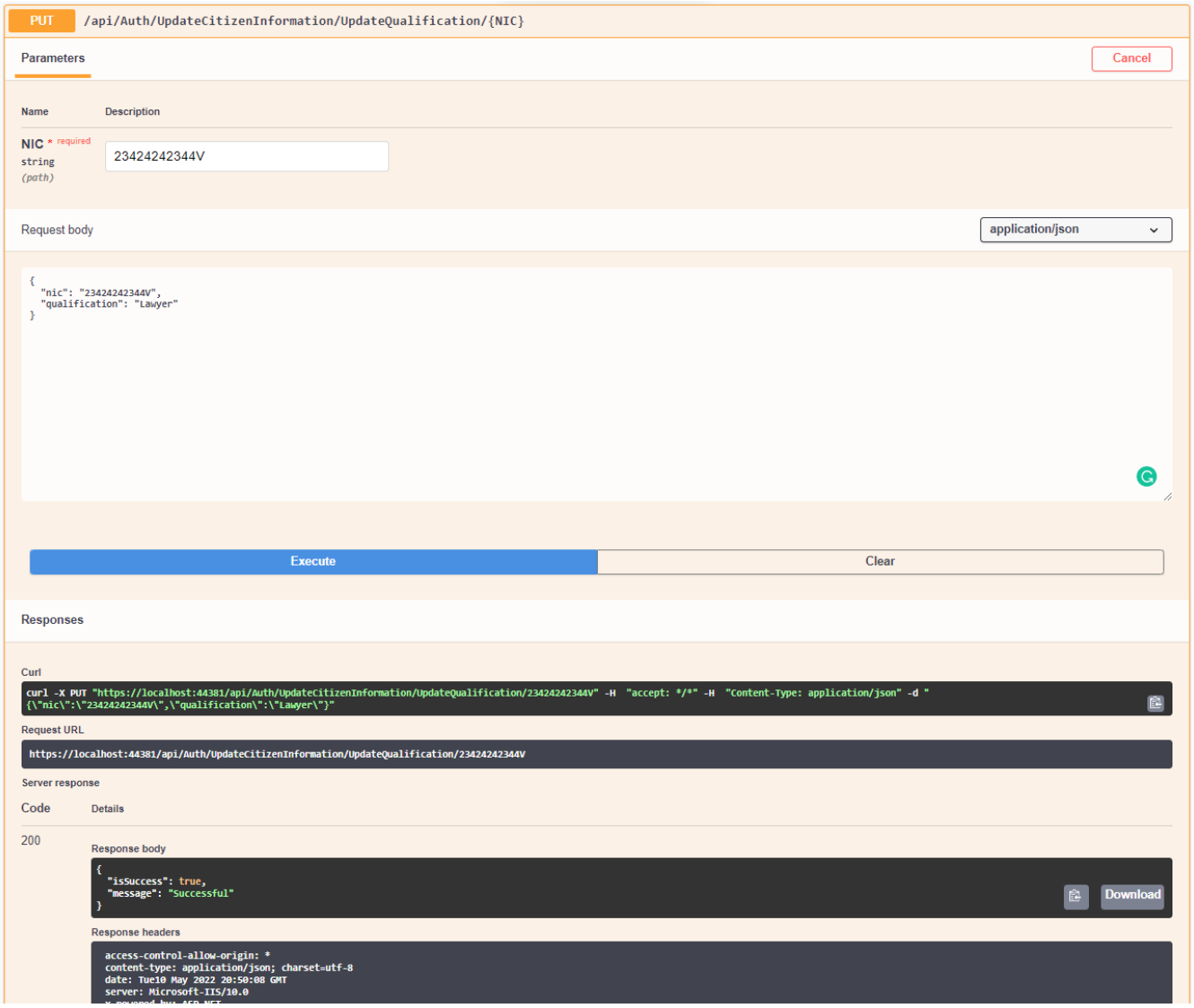


Figure 42 - Run-Time: Update Citizen Qualification by NIC with JSON result

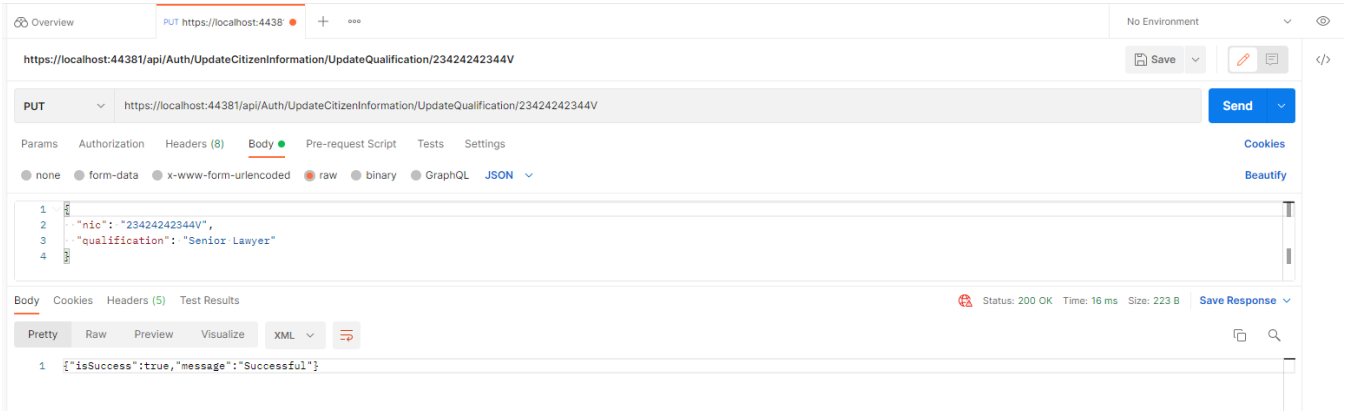


Figure 43 - Run-Time: Update Citizen Qualification by NIC with XML result

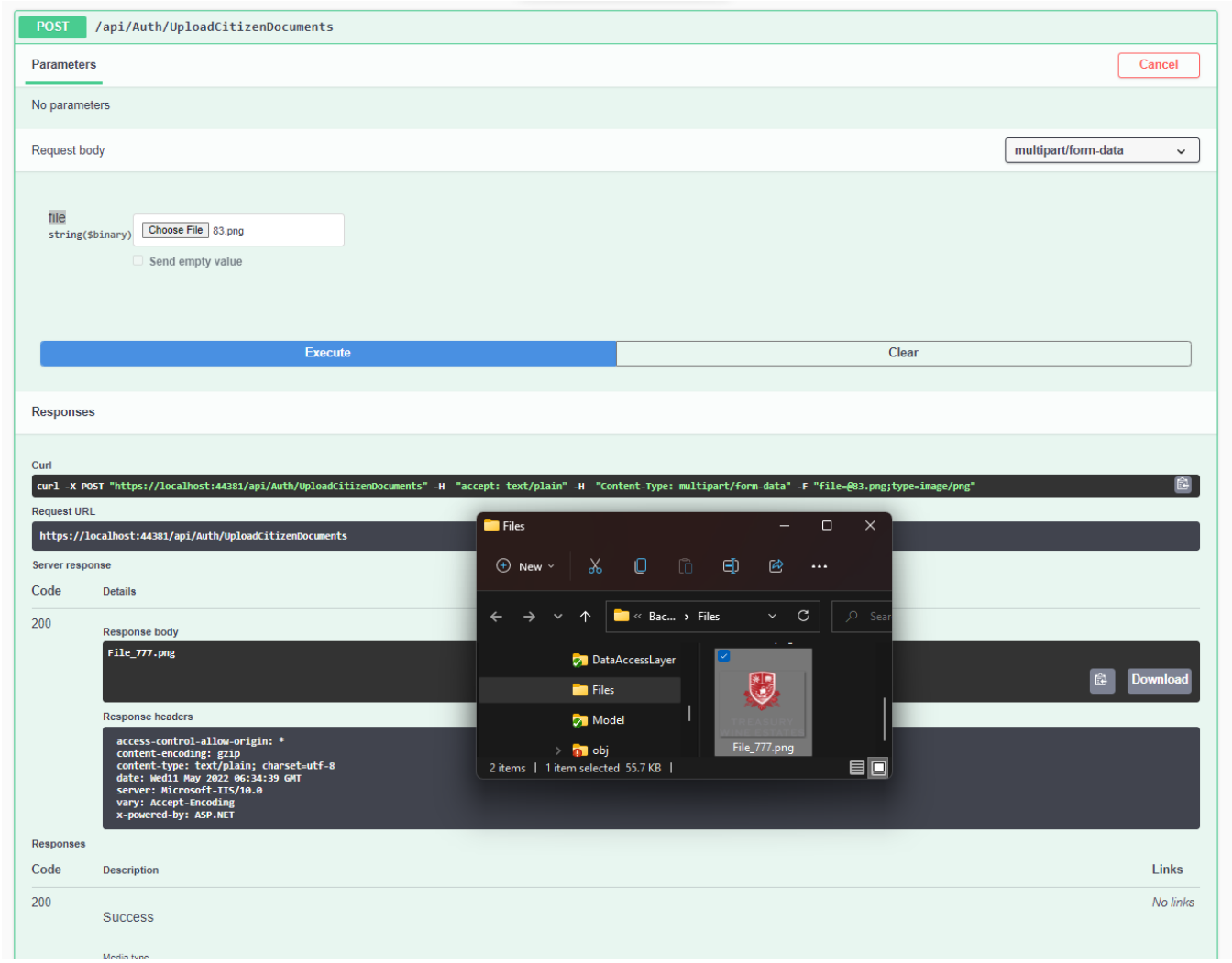


Figure 44 - Run-Time: Upload Citizen Documents with JSON result

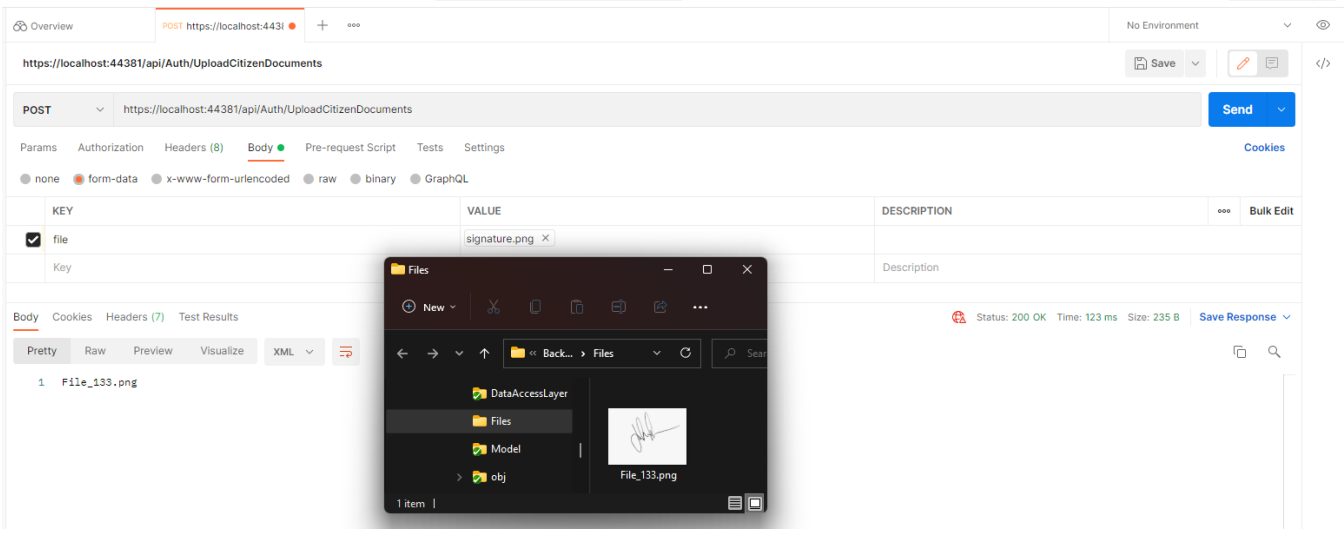


Figure 45 - Run-Time: Upload Citizen Documents with XML result

Create a Complaint by Citizen

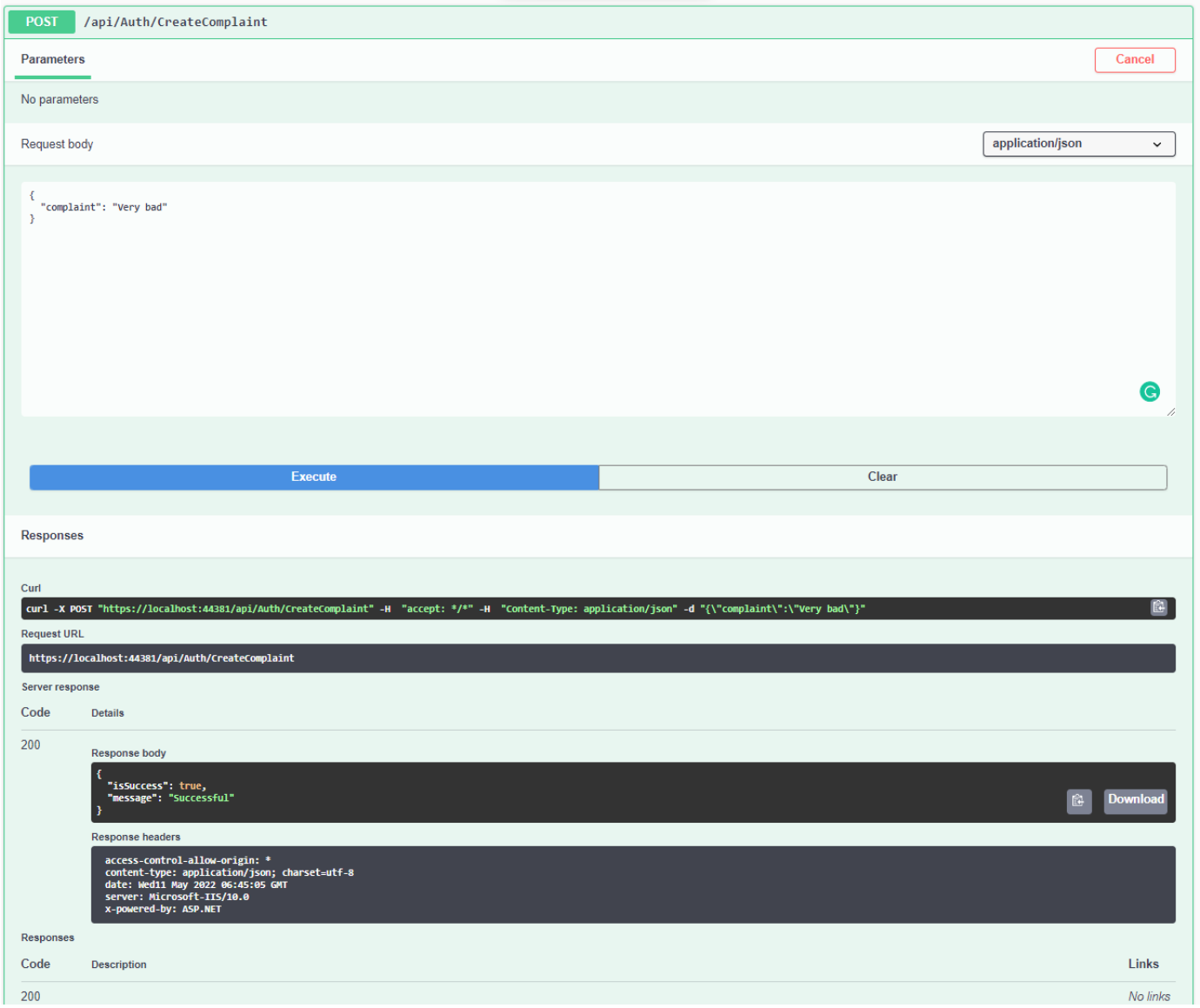


Figure 46 - Run-Time: Create a Complaint by Citizen with JSON result

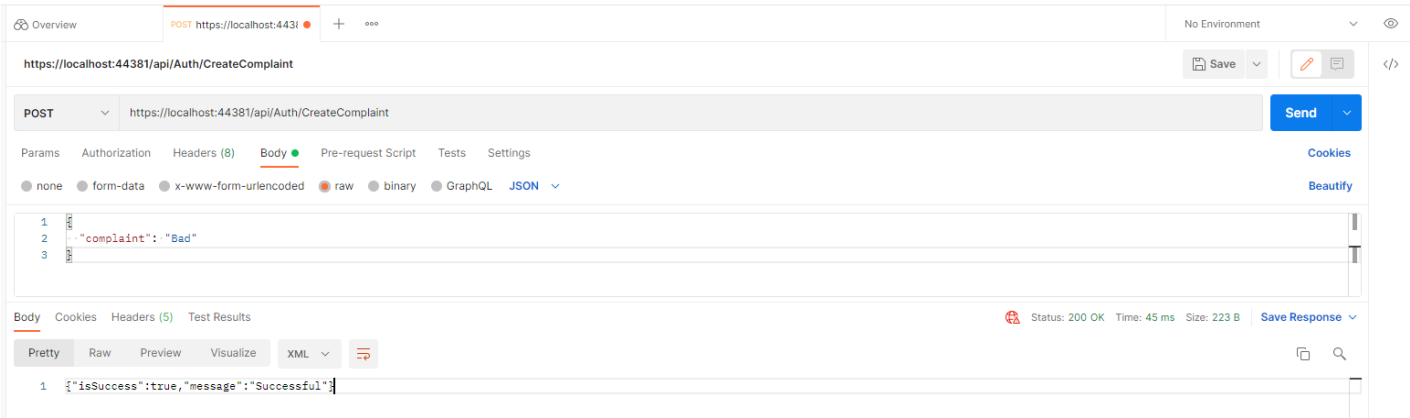


Figure 47 - Run-Time: Create a Complaint by Citizen with XML result

Get List of Complaints

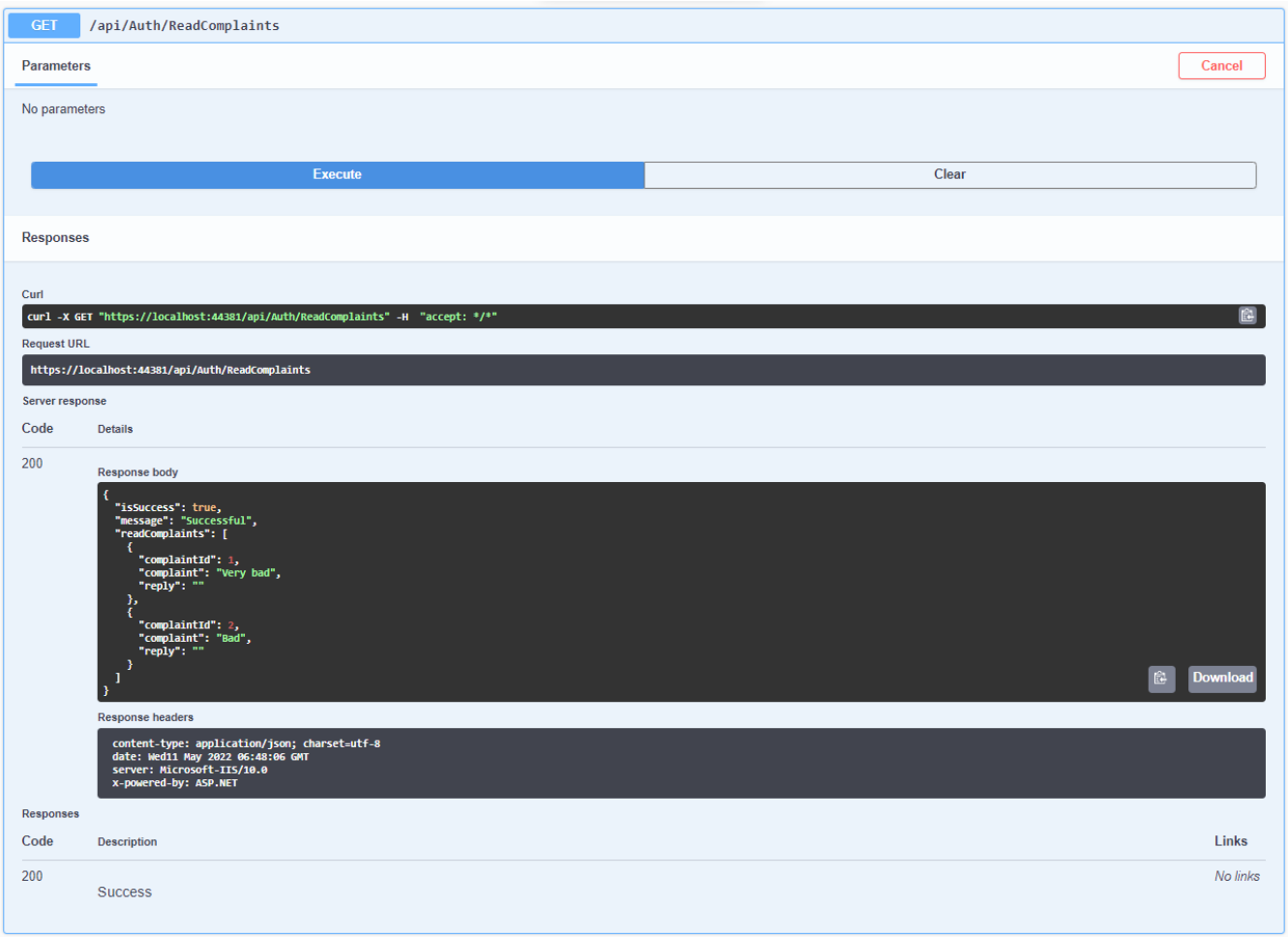


Figure 48 - Run-Time: Get List of Complaints with JSON result

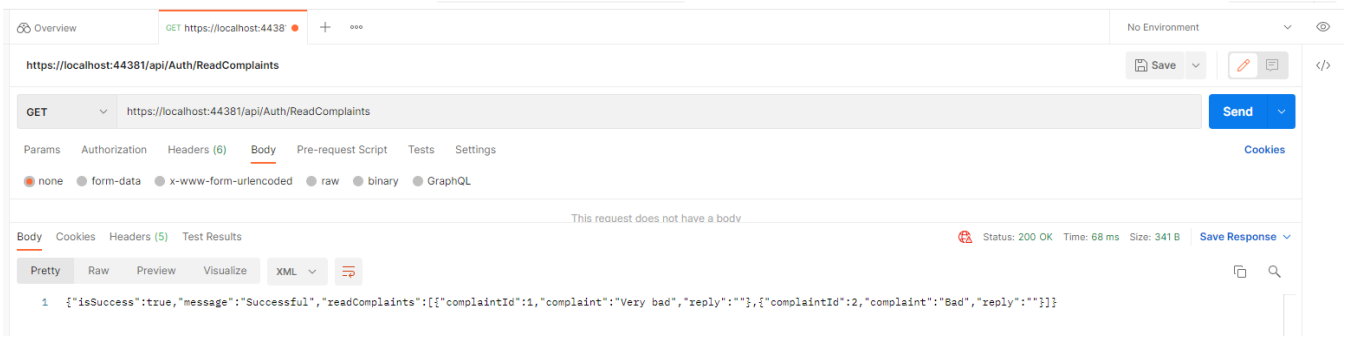


Figure 49 - Run-Time: Get List of Complaints with XML result

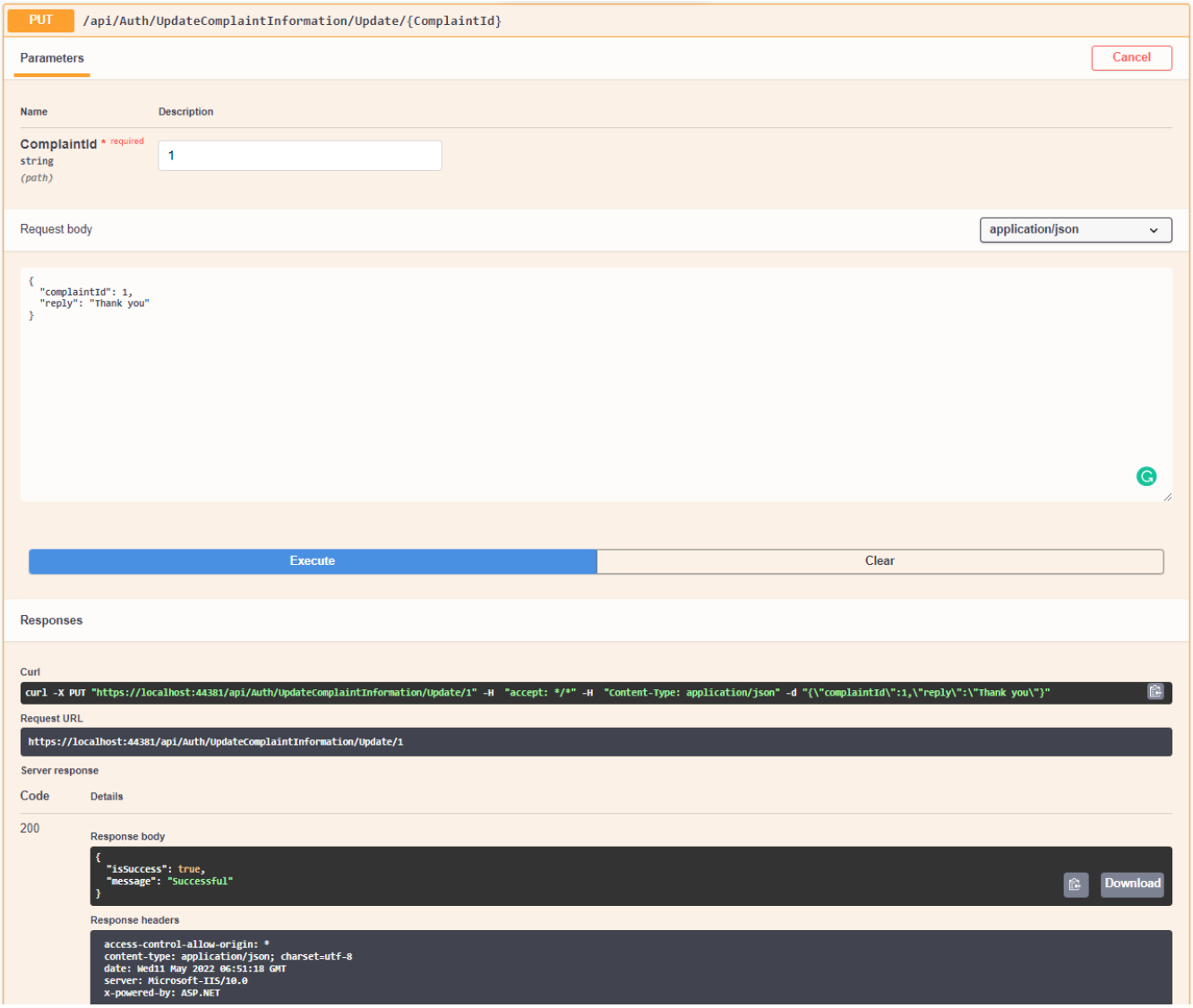


Figure 50 - Run-Time: Reply to the Citizen's Complaint with JSON result

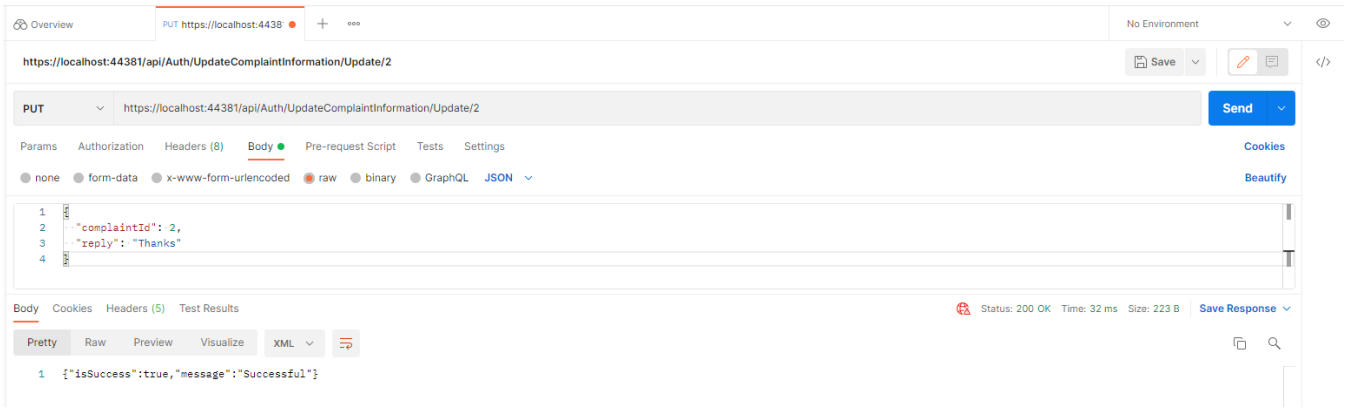


Figure 51 - Run-Time: Reply to the Citizen's Complaint with XML result

2.2.2.2. Mobile Application

Sign Up Forms => Citizen and Officer

Figure 52-Sign-Up

Sign In forms => Citizen and Officer

Figure 53-Sign In

- When new user comes into the system he must be register to the system. They can choose either option from Citizen and Officer and signup with the system. They should fill all the fields (NIC, Name, Age, Address, Profession, Email, password and confirm password and check the radio button called. Then new user can press the sign-up button and able to be a new member of SLBFE company.

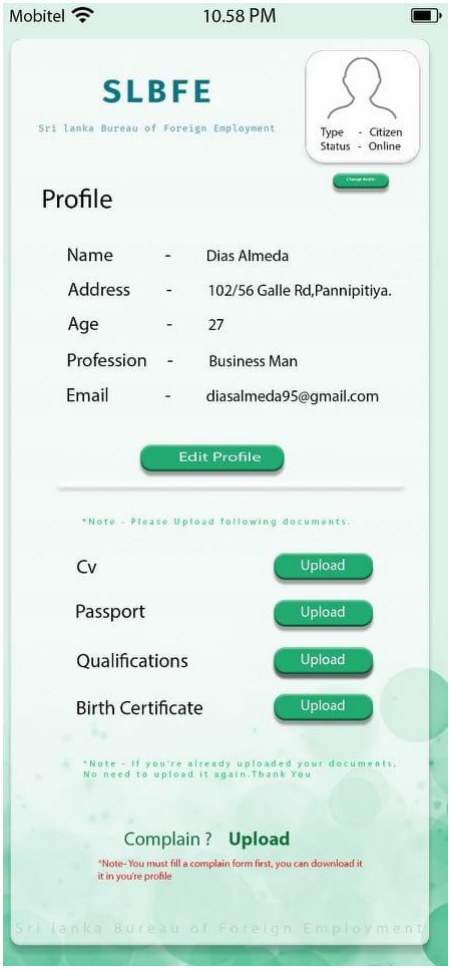
- After signing up with the system whether the user is a citizen, officer, or admin they will be redirected to their own signing pages where they must enter only the email and password. They must enter correct credentials to log in.

Sign Up Form => Citizen

Figure 54-Sign Up Citizen

Dashboard / Home page => Citizen

Figure 55-Sign Up Citizen with data

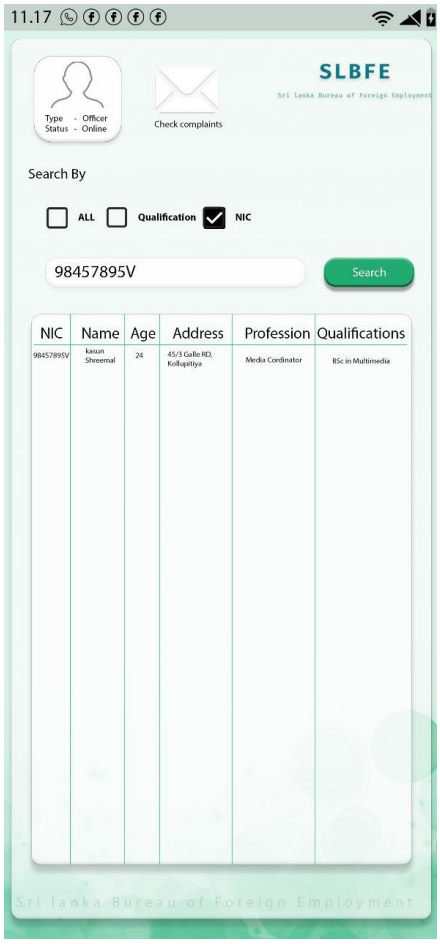


- When a citizen signup to the system their credentials will be sent to the database through an API and check whether there is an existing user with those credentials. If not, the user will be signed up with the system and then have to login with the previously entered details, which will call the APIs to check whether they have entered correct credentials. In the home page the citizen can edit their details and upload any documents they want through the application. They also can make a complaint by uploading a complaint. All these functions are sent to the database through APIs.

Home Page / Dashboard => Officer

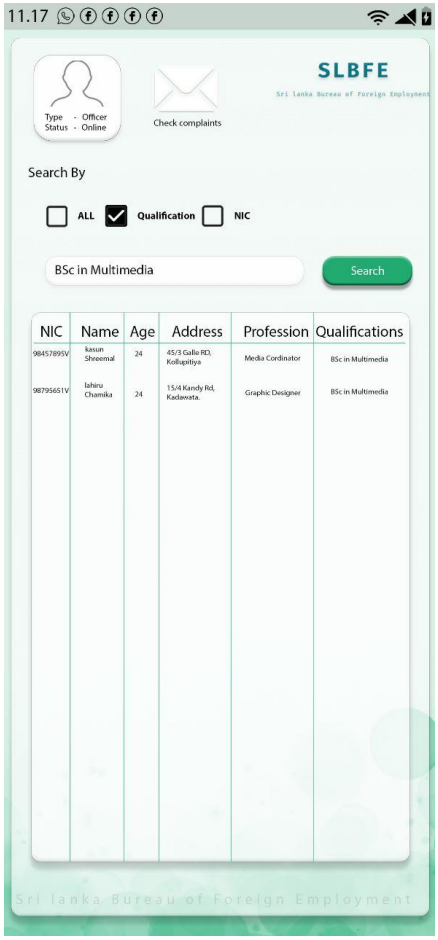
Selecting by NIC

Figure 56-Selecting by NIC



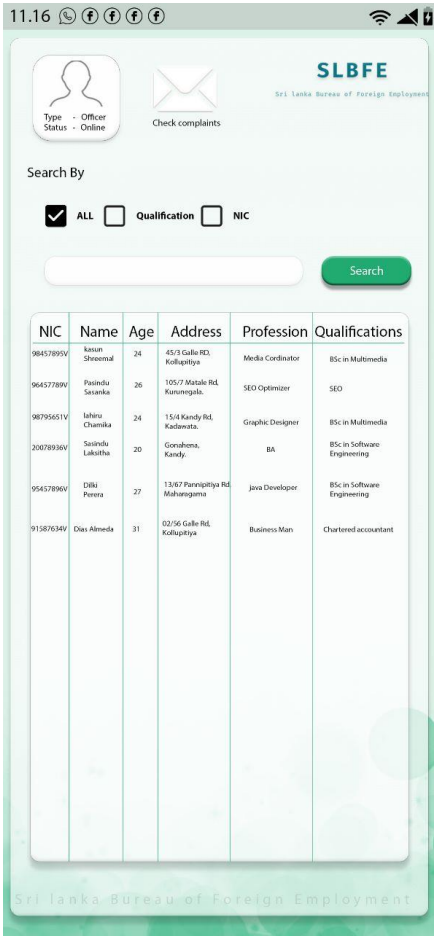
Selecting by Qualification

Figure 57-Selecting by qualification



Selection of all records

Figure 58-Selecting by all records

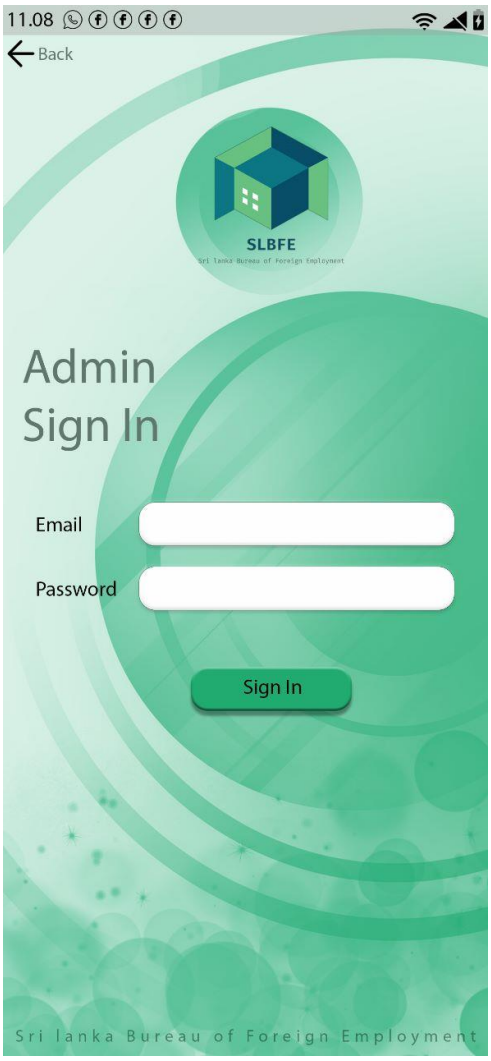


- By selecting Officer's radio button in the signup form the signup details of the officer will be shown and when officer signup to the system their credentials will be sent to the database through an API and check whether there is an existing user with those credentials. If not, the officer will be signed up with the system and then have to login with the previously entered details, which will call the APIs to check whether they have entered correct credentials.

- In the above screenshots by selecting a query and searching them in a search bar panel the API will request to check whether the entered value is there in the body and return the response values with the entered search items. By ticking NIC and searching a NIC number if citizen details are there with the searched NIC number the results will be shown in the table. Like that by ticking the box the responses will be shown in the table.

Sign Up => Admin

Figure 59-Sign-In Admin



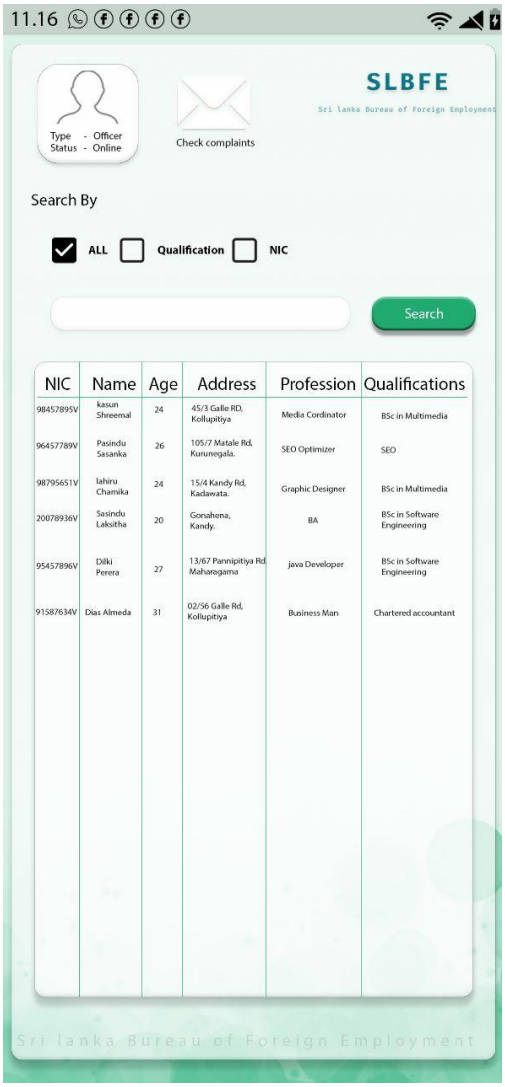
Home Page => Home Page / Dashboard

Figure 27-Admin home

- The admin has its own email and password so a new admin cannot register with the system. When the admin enters its credentials, a request will be sent through API to check whether there is an admin with that credentials. So, if the credentials match, he will be redirected to the system. The above screenshots show the login and dashboard of the admin.

Selecting all records

Figure 60- Selecting All data



Selecting by Email and Address

Figure 28-Selecting Email & Address, delete

- This is the dashboard of the admin and in the above screenshots by selecting a query and searching them in a search bar panel the API will request to check whether the entered value is there in the body and return the response values with the entered search items. By selecting all, all the records in the body will be shown in the table by sending a request through API and the responses will be given according to the request. So, the admin can view the responses given by the APIs. Also, they can delete any account that they think of no longer in use.

3. Tools and Technologies

- When developing the backend of this application we chose ASP .Net web API since this is the backend technology that we are most familiar with and given the limited time frame this let us develop the application faster and easier while resolving the bugs and errors much faster than if we used some other technology.

- The APIs related to the application are deployed and run-on Swagger. The APIs have been tested on Swagger before releasing to the public.

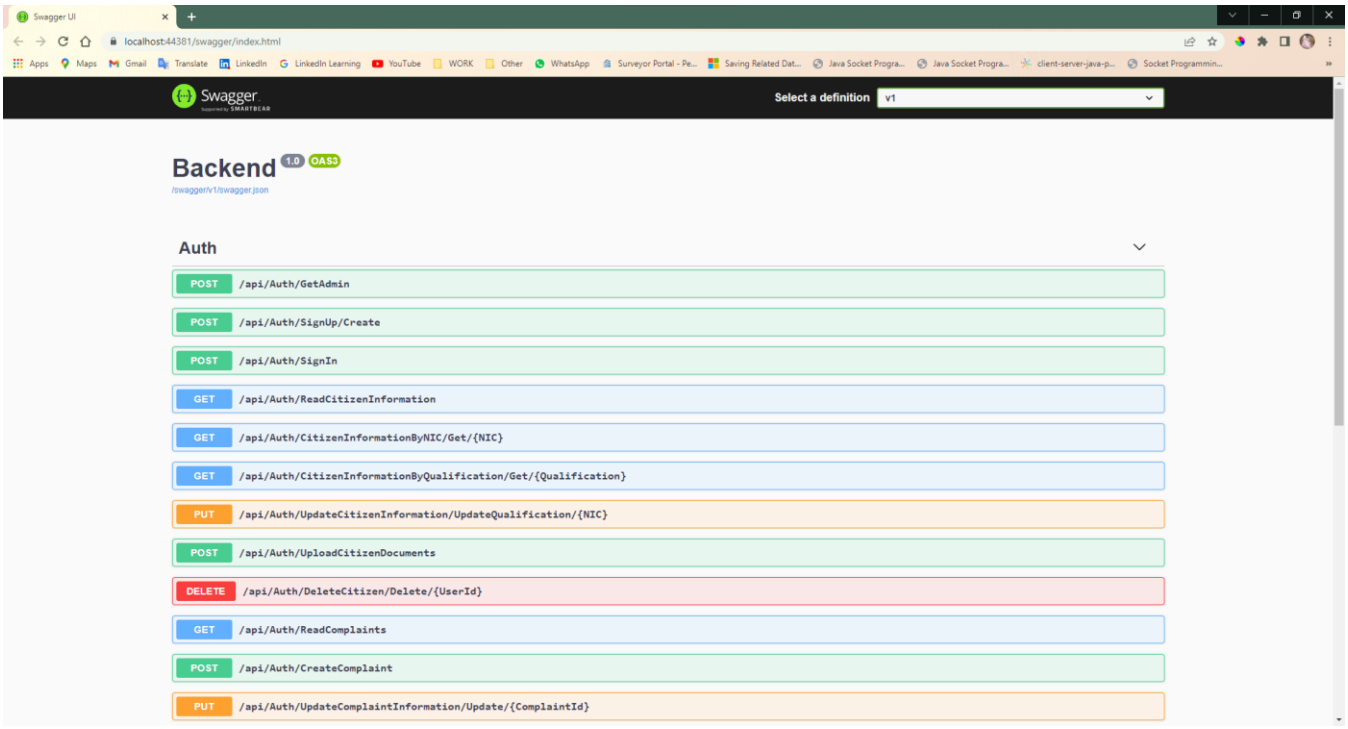


Figure 61 - Swagger API Manager

- The database that was utilized for this application is the MYSQL database.

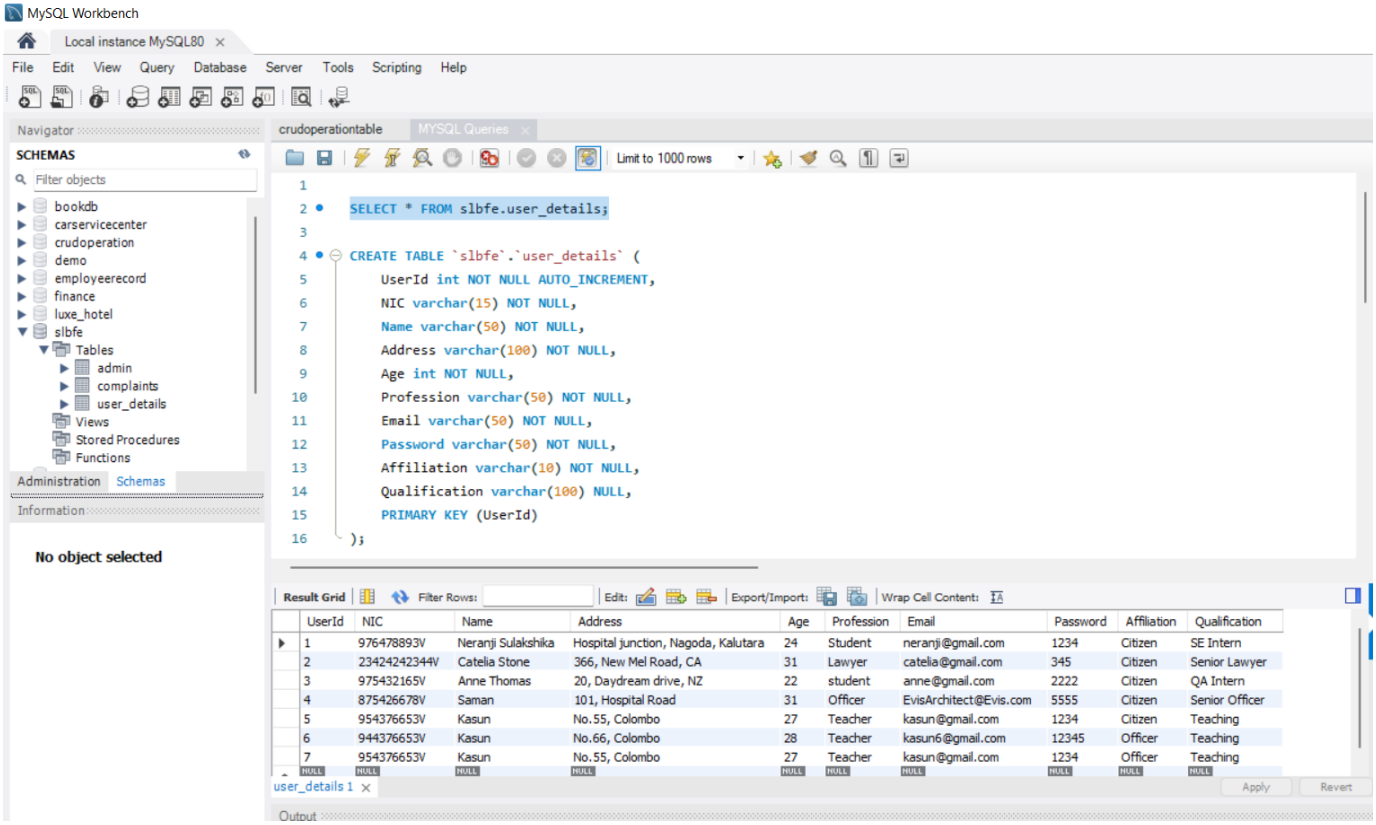


Figure 62 - Execute Screenshot of the user_details table result

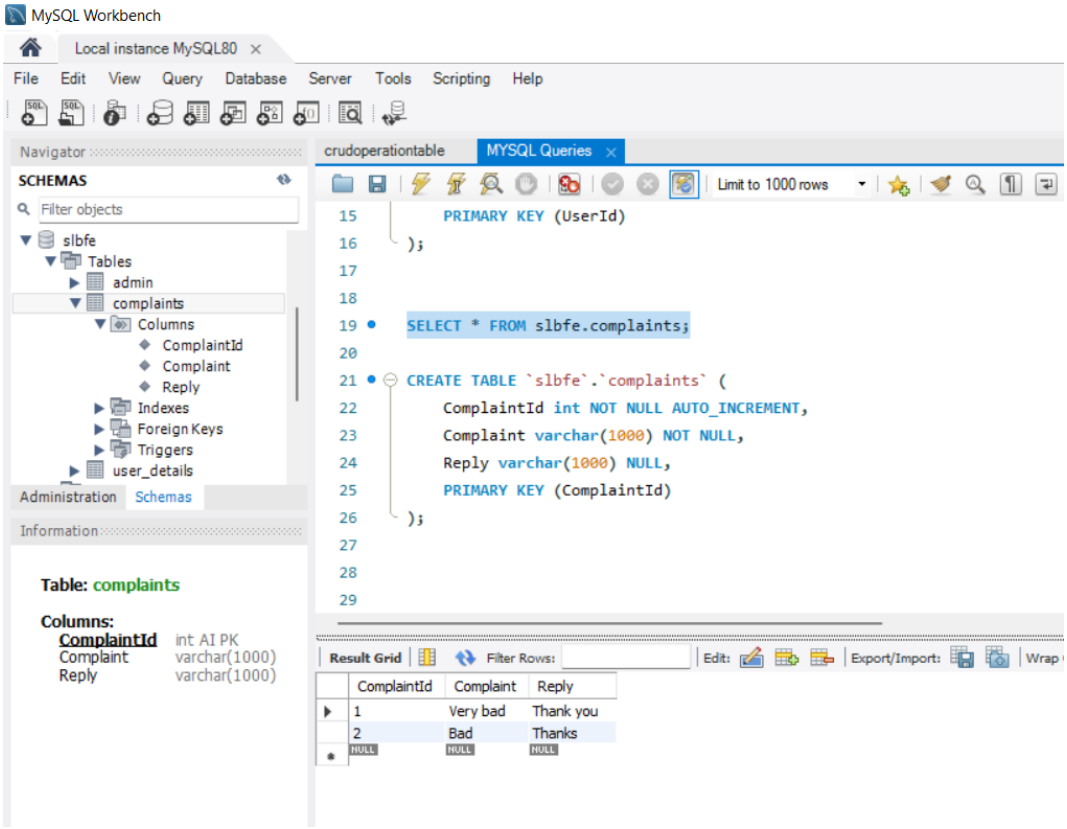


Figure 63 - Execute Screenshot of the complaints table result

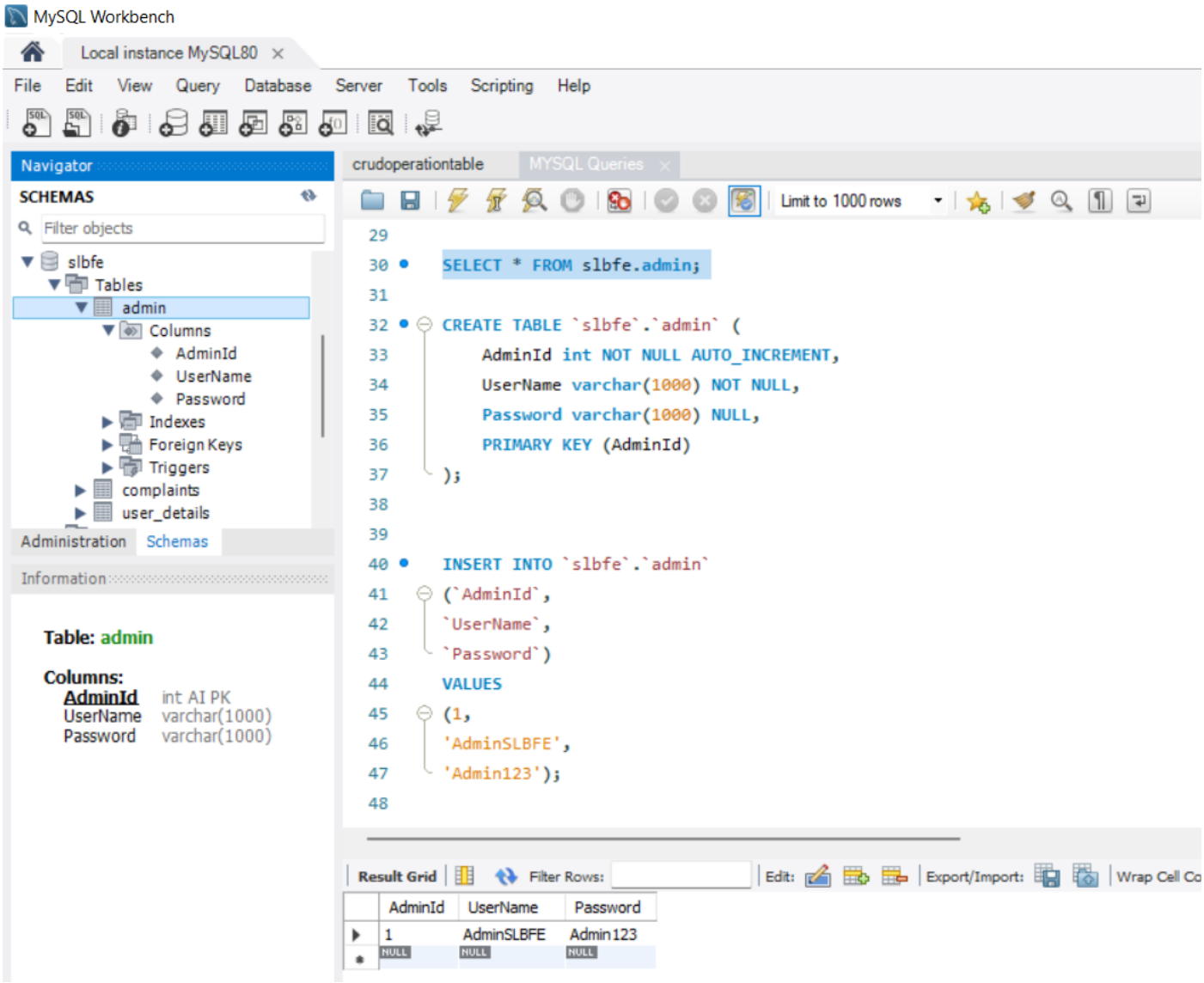


Figure 64 - Execute Screenshot of the admin table result

- Since this is an application developed to help find jobs by having both mobile and web applications, they can reach a high audience as anyone can access either via web or app. By using a mobile app, they can access to the app easily and the functions would run smoothly. Therefore, a mobile application was created to cater more of SLBFE customers.

3.1. Platform - Web Platform

- A web application was created in order to access the system through the web for those who prefer to use this application in their computer.
- When developing the SLBFE web application we utilized ASP .NET in developing the backend, Swagger for managing the implemented APIs.
- For the frontend development of the web application, we utilized ReactJS since it is a versatile framework that enable us to develop responsive interfaces.

3.2. Platform – Mobile Platform

- To build the SLBFE mobile application many technologies was used. This application is an android application which was developed in Android Studio IDE. The application was created with Java language and the backend was ASP.NET. And as for the communication of APIs Retrofit was used. Retrofit is a REST client designed for Java and Android which allows to upload and retrieve JSON. By using Retrofit it is easy to use APIs with the designed user interfaces.

4. Individual Contribution

4.1. 10707385 - V P N Sulakshika

I was responsible for implementing the complete backend of the application with all the API configurations. This consists of the ‘get admin’ which is capable of retrieving the admin profile when logging into the system, ‘create’ is capable of creating the citizen and officer profiles, ‘Sign-In is responsible for retrieving the citizen and officer profiles when logging into the system, ‘read citizen information’ is an API which enables the admin and officers to read the details of the citizens, ’Citizen Information by NIC’ capable of retrieving the details of particular citizens by providing the NIC number of that citizen, ‘Citizen Information by Qualification’ enables the officers to read the details of the citizens with regards to their qualifications. The officers are able to find candidates based on their qualifications with this API. ‘Update Qualification’ enables the citizens to update their qualifications after the creation of their profile. ‘Upload Citizen Documents’ enables the citizens to upload their documents to the systems, ‘Delete Citizen’ can be used by the admin to disable the accounts of the citizens, ‘Create Complaint’ enables the citizens to generate and submit complaints, ‘Read Complaints’ enables the officers and citizens to view the complaints posted by the citizens. ‘Update Complaint Information’ enables the offers to reply to the complaints posted by the citizens.

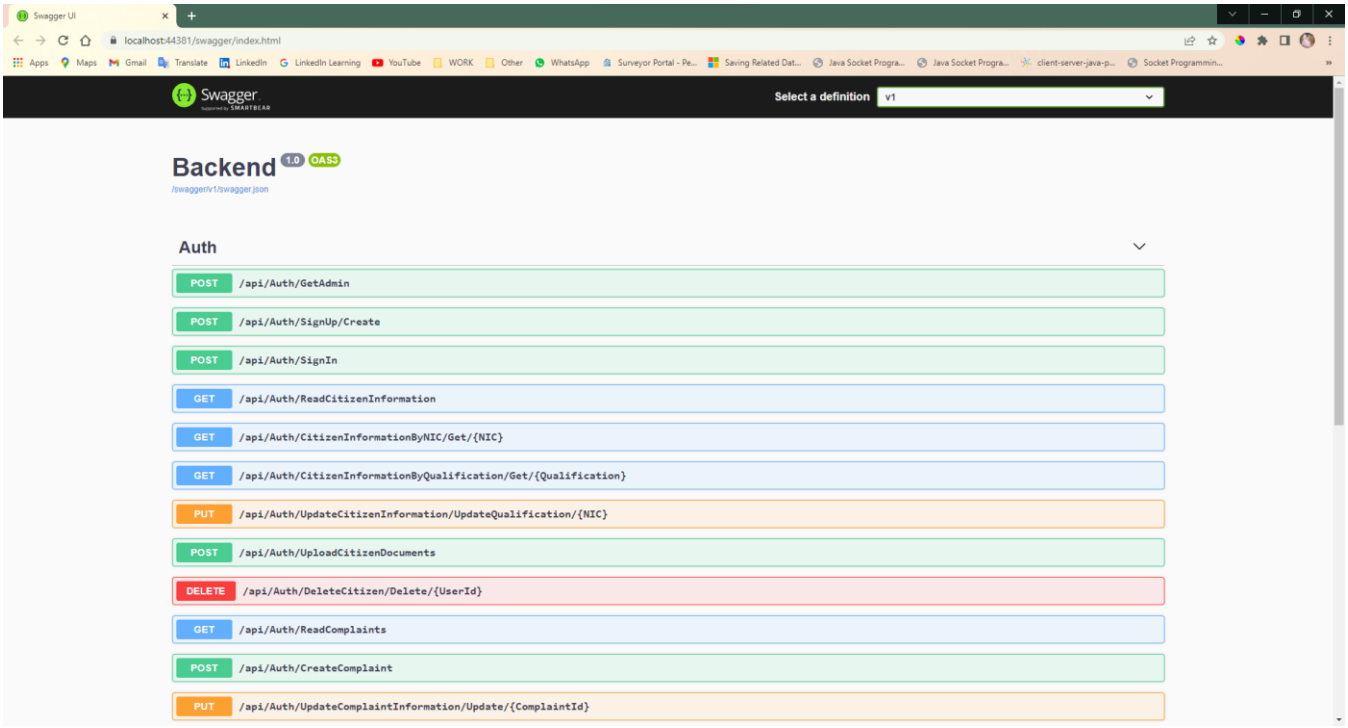


Figure 65 - The Full Backend API Part

As for the frontend development, I was responsible for developing the citizen Sign-In page, officer Sign-In page, citizen signup page, and officer signup page for the roles of the citizen and the officers. I was also responsible for developing the Sign-In page for the admin, the list of citizens the admin could view, and the citizen deletes function for the admin. And also, these all fields are validated.

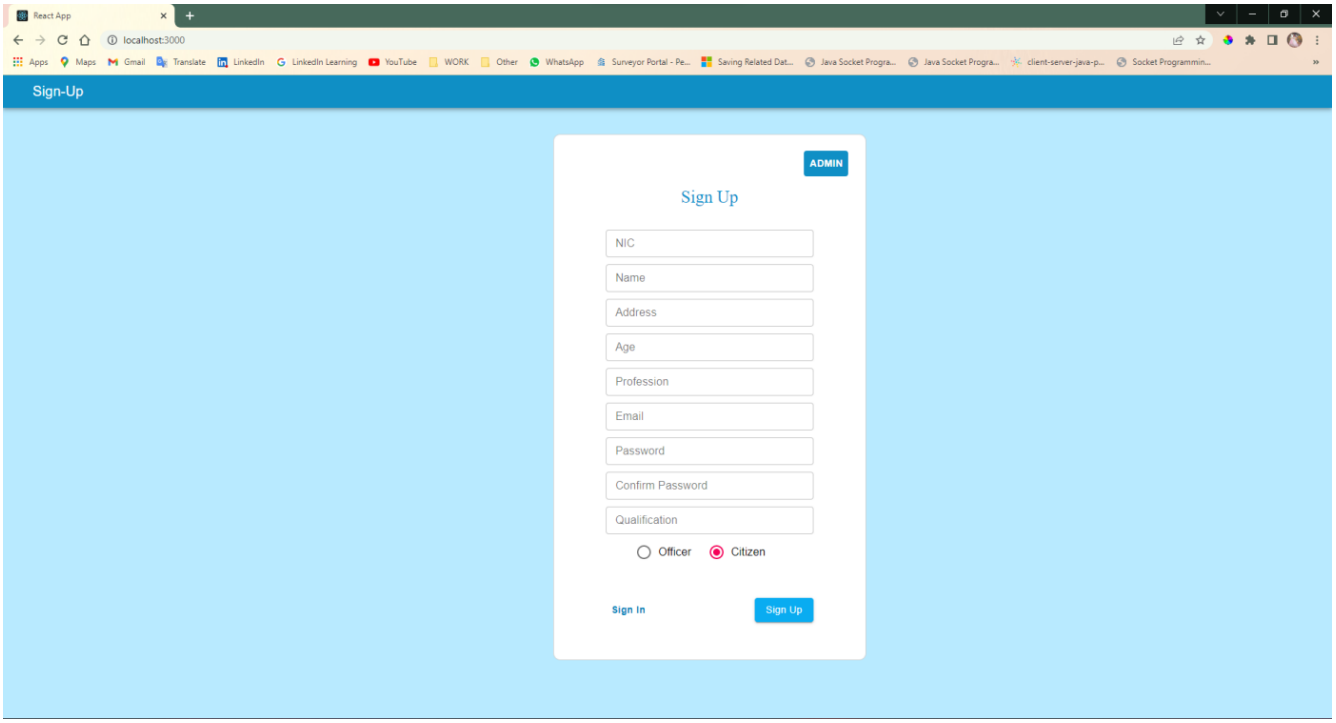


Figure 66- The Frontend Part (Citizen & Office: SignUp)

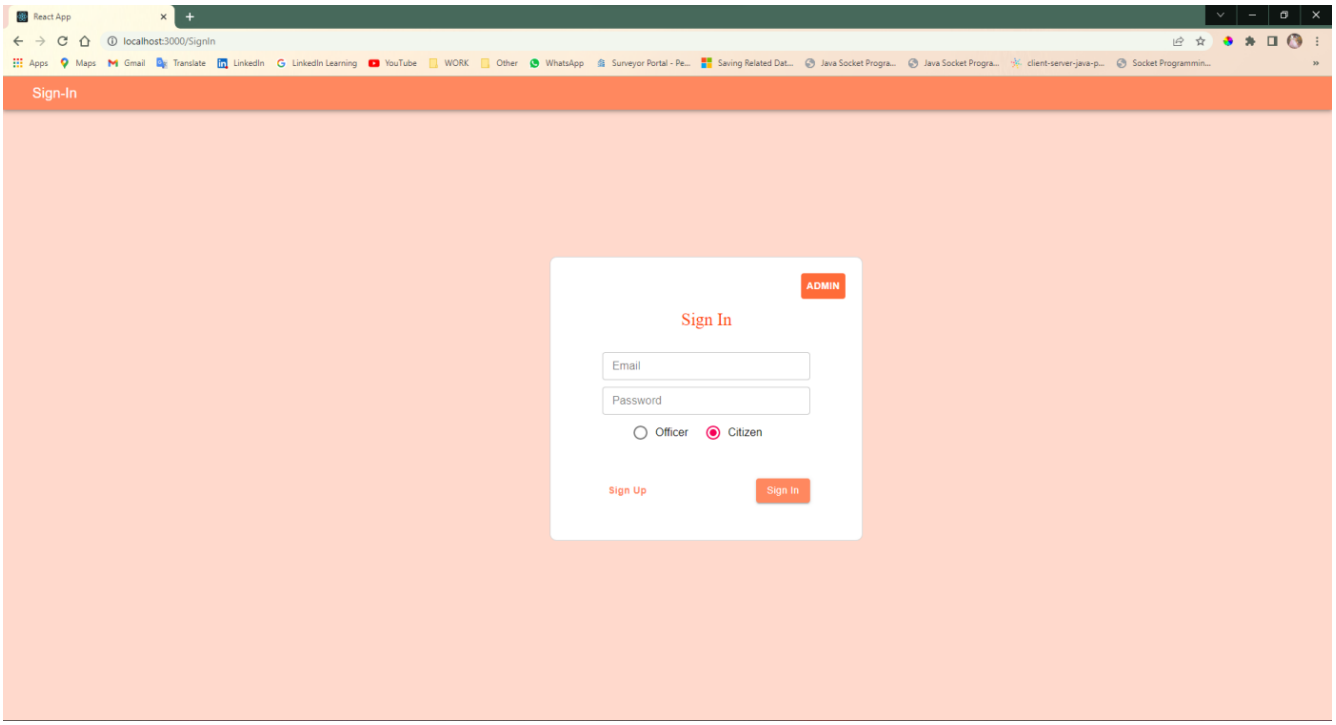


Figure 67 - The Frontend Part (Citizen & Office: SignIn)

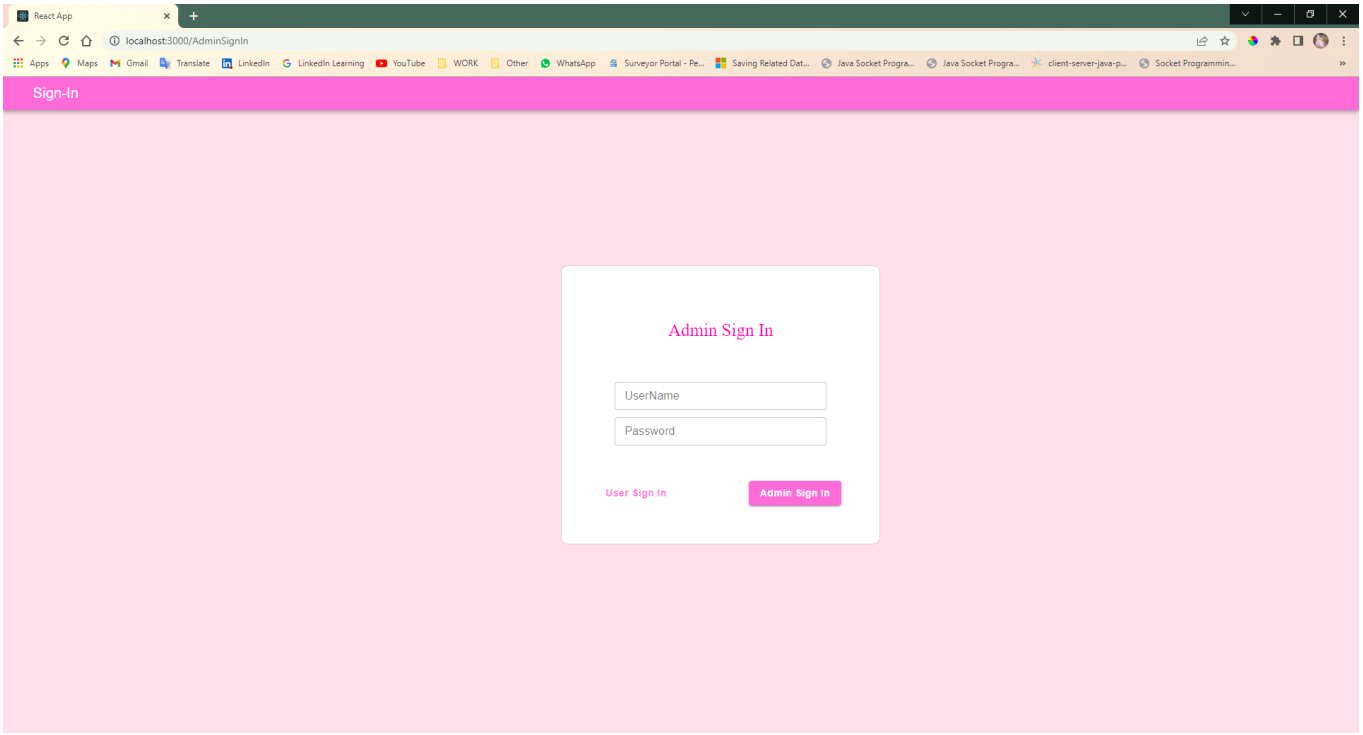


Figure 68 - The Frontend Part (Admin SignIn)

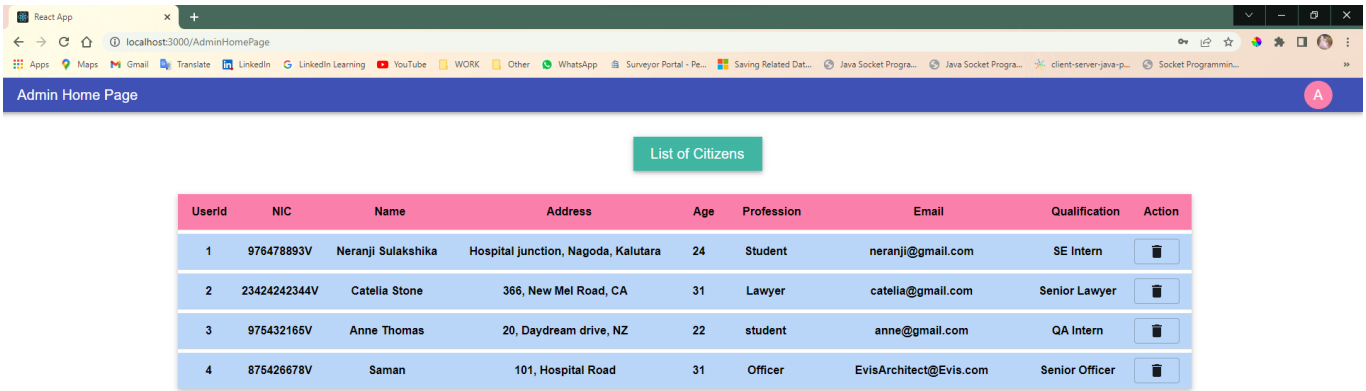
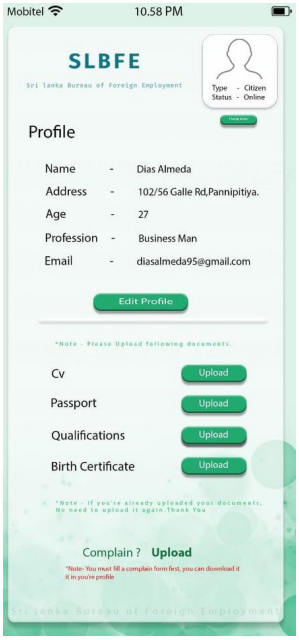


Figure 69 - The Frontend Part (Admin Home Page with View & Delete Citizens' Information)

4.2. 10707241 - H S Kaushalya

I made the mobile application with my other two colleagues. There I made the registration and homepage of the citizen along with its backend. I used retrofit 2 service for the API data communication all user input data are handle by the APIs and I use POST method for the user registration, from that method all the user's credentials are store to the data base by POST API and I used GET API for the signing process. In that method when user input his username and password first it gets all the usernames in the table and check the username is available or not in the database if it is available then it gets the password which belongs to that username and then check whether the user input password is same to the password which on database if those are correct then user can login to the system successfully. Then I created the home page of the citizen where they can update their details like address, age, qualifications, and other documents. So, to update the details I used the PUT API. Also, the user can create a new complaint for that I used the POST API. So, these are the interfaces that I created and joined the backend to them using Retrofit and APIs.



4.3. 10707417 – H J K I Wijerama

I created Admin Sign-In page. I used GET API for the sign in process of Admin. Admin can sign up by entering their username and email. Then that data check with the database and if they are available admin can sign in successfully. If the data in the database doesn't match with the entered data, admin is not able to log into the system.

I was also responsible for developing the home page for the admin. It includes the list of citizens that admin can view by filtering their contacts details like email and address. For that function I used GET API for get all the requested data filtering by email and the address from the database and display them in the mobile application. And also, admin have the delete function to delete any citizen. For the delete function I used DELETE API. All the data should retrieve from database.

4.4. 10707129 – K L D Anupama

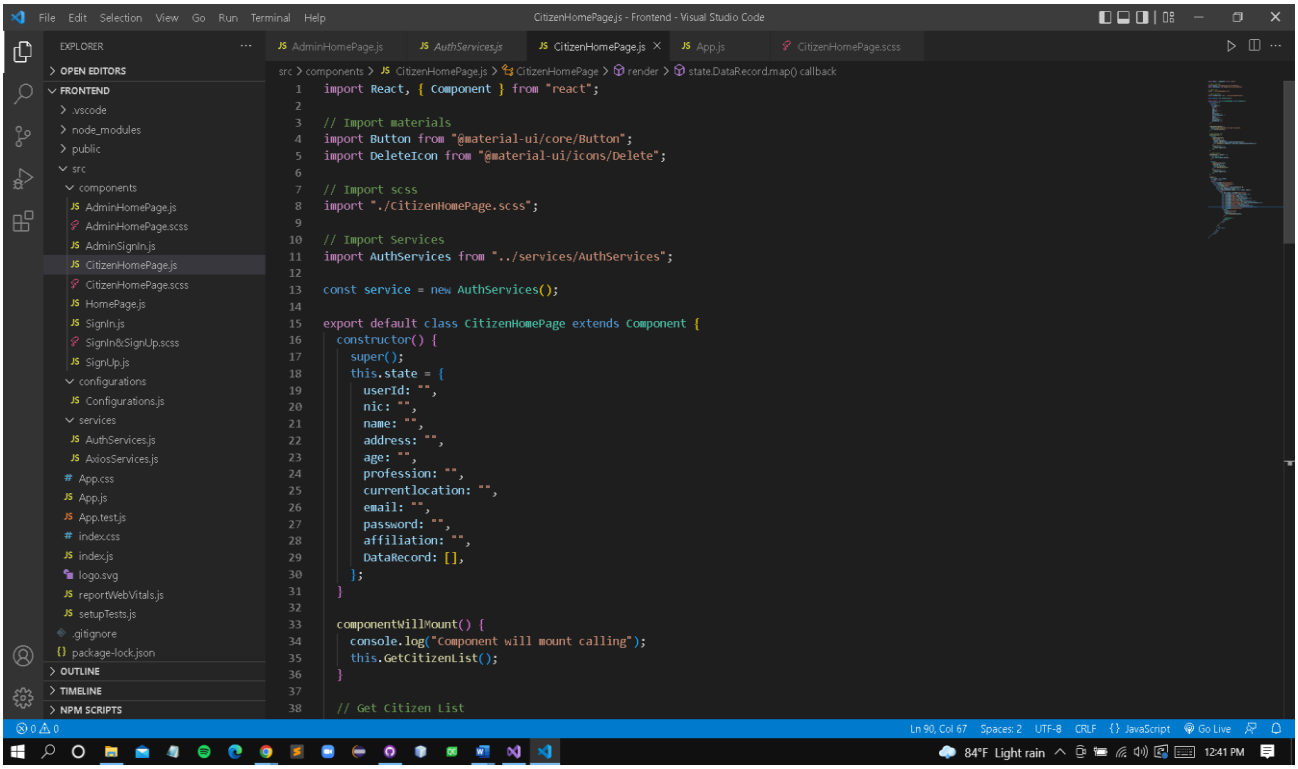
As for the frontend development of mobile application, I was responsible for developing the Officer Sign-In page, officer Sign-up page. I was also responsible for developing the home page for the officer, the list of citizens filtering by NIC of citizens and the Qualifications of citizens. To check the data of citizens by NIC and the qualifications, GET API is used. And also, officers can check complaints. I created the complaint reply interface to view reply to complaints by GET API. All the data should retrieve from database.

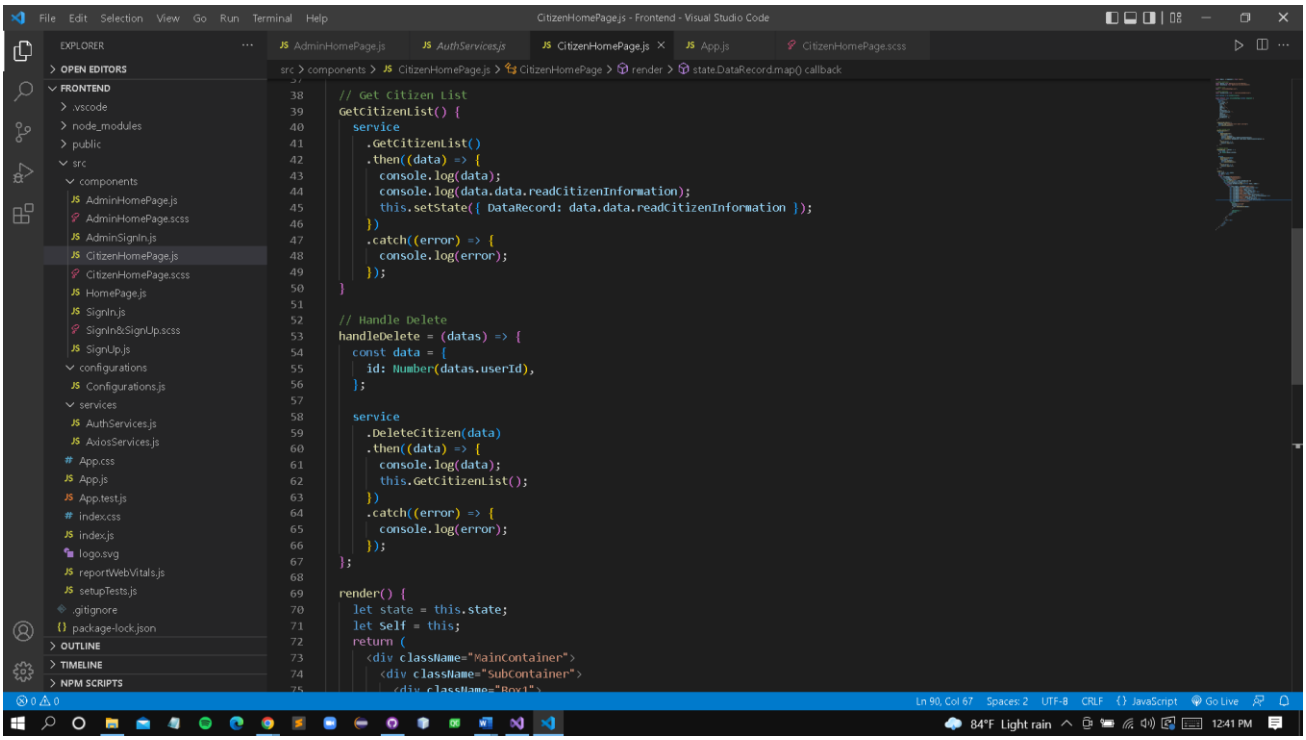
All the officer credentials are store to the database by POST API and I used to retrofit 2 service for the API communication. All officers' inputs and outputs are handled by APIs. From POST method all officers' inputs store into the database and I used GET API for sign in process. From that method, when officer enter their credentials, then check from the database availability and the accuracy of the data. If the password and the username of the given officer is available officer can successfully log into the system.

4.5. 10707372 - P D S Sigera

In this coursework my contribution goes out, as to creating the citizens homepage frontend part. In the registration form the citizens have to fill out their personal information such as NIC, Name, Address, Age, Profession, Current Location, Email, Password, Affiliation. There are two functions for this citizen frontend part, they are POST method and PUT method. In the POST method is to update the data on the server and the PUT method is to add new data on the server. In the POST method the citizen can upload their birth certificates, their CVs and their passport copies. In the PUT method the citizen can update their current location (longitude and latitude) and also make a complaint if necessary. I have used React JS in implementing the code of the frontend and for the database I have used MYSQL Workbench in implementing the database.

Screen shots of the code Citizen Home Page



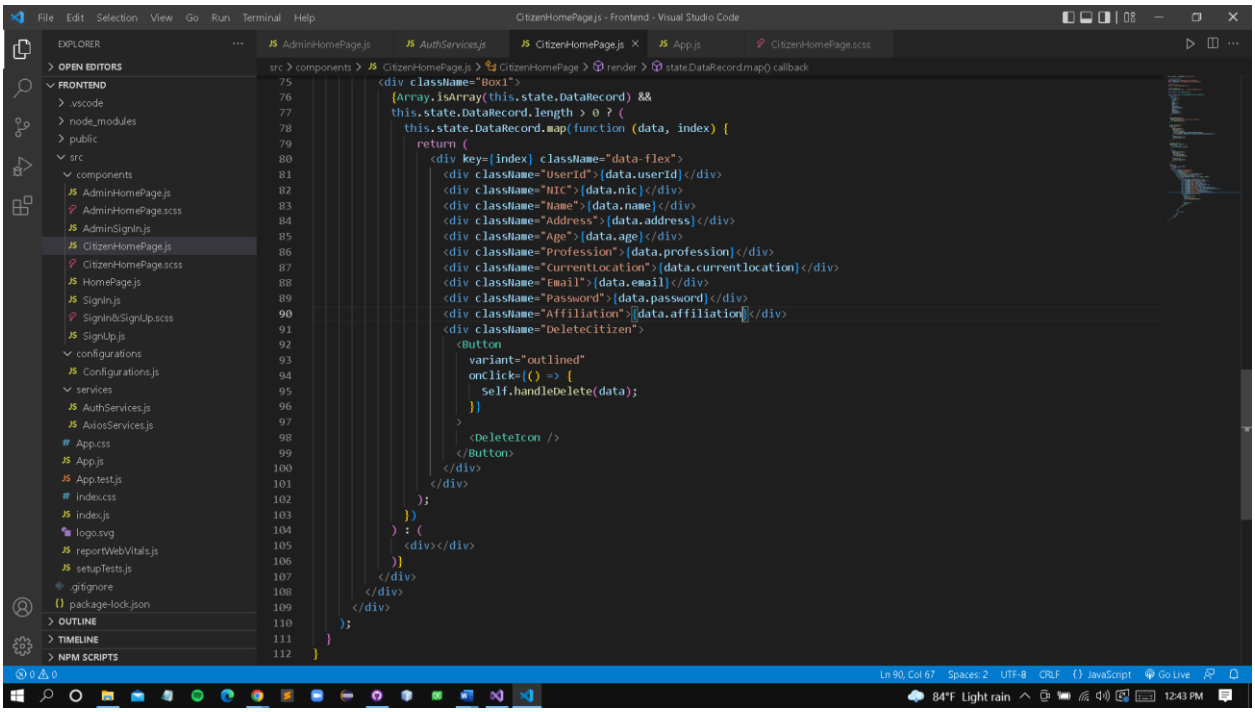


```
// Get Citizen List
GetCitizenList() {
  service
    .GetCitizenList()
    .then((data) => {
      console.log(data);
      console.log(data.data.readCitizenInformation);
      this.setState({ DataRecord: data.data.readCitizenInformation });
    })
    .catch((error) => {
      console.log(error);
    });
}

// Handle Delete
handleDelete = (datas) => {
  const data = {
    id: Number(datas.userId),
  };

  service
    .DeleteCitizen(data)
    .then((data) => {
      console.log(data);
      this.GetCitizenList();
    })
    .catch((error) => {
      console.log(error);
    });
};

render() {
  let state = this.state;
  let Self = this;
  return (
    <div className="MainContainer">
      <div className="SubContainer">
```



```
          <div className="Box1">
            {Array.isArray(this.state.DataRecord) &&
            this.state.DataRecord.length > 0 ? (
              this.state.DataRecord.map(function (data, index) {
                return (
                  <div key={index} className="data-flex">
                    <div className="UserId">{data.userId}</div>
                    <div className="NIC">{data.nic}</div>
                    <div className="Name">{data.name}</div>
                    <div className="Address">{data.address}</div>
                    <div className="Age">{data.age}</div>
                    <div className="Profession">{data.profession}</div>
                    <div className="CurrentLocation">{data.currentlocation}</div>
                    <div className="Email">{data.email}</div>
                    <div className="Password">{data.password}</div>
                    <div className="Affiliation">{data.affiliation}</div>
                    <div className="DeleteCitizen">
                      <Button
                        variant="outlined"
                        onClick={() => {
                          Self.handleDelete(data);
                        }}
                      >
                        <DeleteIcon />
                      </Button>
                    </div>
                  </div>
                );
              })
            ) : (
              <div></div>
            )}
          </div>
        </div>
      </div>
    );
  }
}
```

4.6. 10707291 – S S N S Nevins

In this, my contribution has been the officers' part in the front end. Firstly, the officers have to get registered as well, and they do so by entering the same details as the normal citizens. Afterwards, they can either view and validate the information provided by the job seekers or see and reply to their complaints. These are done by the POST and PUT methods relatively. The officers can also get the citizen's details by their NIC's and that is thanks to the GET LIST function. They can use GET method to find candidates based on specific qualifications.

5. Certificates

- Each and every member followed the course "Introduction to Web APIs by Andrew Probert" in LinkedIn Learning Center and we attach the certificate here. (Probert, 2019)

5.1. 10707385 - V P N Sulakshika

Figure 70 - LinkedIn certificate: 10707385

5.2. 10707241 - H S Kaushalya

Figure 71 - LinkedIn certificate: 10707241

5.3. 10707417 – H J K I Wijerama

Figure 72 - LinkedIn certificate: 10707417

5.4. 10707129 – K L D Anupama

Figure 73 - LinkedIn certificate: 10707129

5.5. 10707372 - P D S Sigera

Figure 74 - LinkedIn certificate: 10707372

5.6. 10707291 – S S N S Nevins

Figure 75 - LinkedIn certificate: 10707291

6. Risks and Outcomes

- We weren’t familiar with Node.js and its related technologies that we used, thereby taking more time than we initially thought in order to get used to it etc.

- The constant power outages were a hassle to us as well since it impeded certain tasks (for example ones that required multiple team members).

- Due to the previous risks, we became hard up for time as well. However, the tasks got completed.

7. Summary

The website consists of a backend written in Java and frontend of React.js, while the mobile app uses a .net framework and used android studio to develop it. This system caters to three primary users -the admin can sign in with his/her credentials and the officers of the SLBFE and the citizens who wish to receive foreign employment. This application was developed by using APIs and given guidelines.

8. Conclusion

In conclusion overall, we worked together as a team in developing this project. We were able to learn more about API's and expand our knowledge to Node.js, React and more technologies that were new to us previously, so we had to explore more on how to develop with ReactJS and we watched many tutorials. These are for the web application development. For the mobile application we have developed using android studio. Finally, this new system can replace the outdated one that was being used by the SLBFE as it is current and more useful; thereby fulfilling its requirements.

9. References

Probert, Andrew. “Filter Response with Parameters - REST Video Tutorial | LinkedIn Learning, Formerly Lynda.com.” *LinkedIn*, LinkedIn Learning, 9 Oct. 2019, www.linkedin.com/learning/introduction-to-web-apis/filter-response-with-parameters?autoplay=true&u=26140778.

Sayings, Famous Quotes &. “Top 17 Quotes about Apis: Famous Quotes & Sayings about Apis.” *Quotestats.com*, quotestats.com/topic/quotes-about-apis/. Accessed 11 May 2022.