

**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 appropriateSci/ps/d:/students/cwkfrontcover/2013/14

Table of Contents

1.	Introduction	7
2.	API Documentation	8
2.1.	List of APIs	8
2.2.	Screenshots	10
2.2.1.	Code Screenshots - Backend	10
2.2.1.2.	Mobile Application.....	33
2.2.2.	Run-Time Screenshots.....	40
3.	Tools and Technologies.....	61
3.1.	Platform - Web Platform.....	64
3.2.	Platform – Mobile Platform.....	64
4.	Individual Contribution	65
4.1.	10707385 - V P N Sulakshika.....	65
4.2.	10707241 - H S Kaushalya.....	68
4.3.	10707417 – H J K I Wijerama	69
4.4.	10707129 – K L D Anupama.....	70
4.5.	10707372 - P D S Sigera.....	71
4.6.	10707291 – S S N S Nevins	73
5.	Certificates.....	74
5.1.	10707385 - V P N Sulakshika.....	74
5.2.	10707241 - H S Kaushalya.....	75
5.3.	10707417 – H J K I Wijerama	75
5.4.	10707129 – K L D Anupama.....	76
5.5.	10707372 - P D S Sigera.....	76
5.6.	10707291 – S S N S Nevins	77
6.	Risks and Outcomes.....	78
7.	Summary	79
8.	Conclusion.....	80
9.	References	81

List of Figures

Figure 1 - Admin Sign-In method in the AuthController.cs	10
Figure 2 - Admin Sign-In method in the AuthDL.cs	11
Figure 3 - Citizen & Officer: Sign-Up method in the AuthController.cs	12
Figure 4- Citizen & Officer: Sign-Up method in the AuthDL.cs	13
Figure 5 - Citizen & Officer: Sign-In in the AuthController.cs	14
Figure 6 - Citizen & Officer: Sign-In method in the AuthDL.cs.....	15
Figure 7 - Read Citizen Information method in the AuthController.cs	16
Figure 8 - Read Citizen Information method in the AuthDL.cs	17
Figure 9 - Citizen Information by NIC method in the AuthController.cs.....	18
Figure 10 - Citizen Information by NIC method in the AuthDL.cs	19
Figure 11 - Citizen Information by Qualification method in the AuthController.cs	20
Figure 12 - Citizen Information by Qualification method in the AuthDL.cs	21
Figure 13 - Update Citizen Information method in the AuthController.cs	22
Figure 14 - Update Citizen Information method in the AuthDL.cs	23
Figure 15 - Upload Citizen Documents method in the AuthController.cs	24
Figure 16 - Delete Citizen method in the AuthController.cs	25
Figure 17 - Delete Citizen method in the AuthDL.cs	26
Figure 18 - Read Complaints method in the AuthController.cs	27
Figure 19 - Read Complaints method in the AuthDL.cs	28
Figure 20 - Create Complaint method in the AuthController.cs	29
Figure 21 - Create Complaint method in the AuthDL.cs	30
Figure 22 - Update Complaint Information method in the AuthController.cs	31
Figure 23 - Update Complaint Information method in the AuthDL.cs	32
Figure 24 - Run-Time: Citizen & Officer Sign-Up User-Interface.....	40
Figure 25 - Run-Time: Citizen & Officer Sign-Up with JSON result.....	41
Figure 26 - Run-Time: Citizen & Officer Sign-Up with XML result	41
Figure 27 - Run-Time: Citizen & Officer Sign-In Use-Interface	42
Figure 28 - Run-Time: Citizen & Officer Sign-In with JSON result	42
Figure 29 - Run-Time: Citizen & Officer Sign-In with XML result.....	43
Figure 30 - Run-Time: Admin Sign-In User-Interface	44
Figure 31 - Run-Time: Admin Sign-In with JSON result	44
Figure 32 - Run-Time: Admin Sign-In with XML result.....	45
Figure 33 - Admin Home User-Interface.....	46
Figure 34 - Run-Time: List of Citizen Information with JSON result	46
Figure 35 - Run-Time: List of Citizen Information with XML result.....	47
Figure 44 - Run-Time: Delete Citizen Account with JSON result	48
Figure 45 - Run-Time: Delete Citizen Account with XML result.....	48
Figure 36 - Run-Time: Get Citizen Information by NIC with JSON result	49
Figure 37 - Run-Time: Get Citizen Information by NIC with XML result.....	49
Figure 38 - Run-Time: Get Citizen Information by Qualification with JSON result	50
Figure 39 - Run-Time: Get Citizen Information by Qualification with XML result	50
Figure 40 - Run-Time: Update Citizen Qualification by NIC with JSON result	51
Figure 41 - Run-Time: Update Citizen Qualification by NIC with XML result.....	51

Figure 42 - Run-Time: Upload Citizen Documents with JSON result	52	
Figure 43 - Run-Time: Upload Citizen Documents with XML result	52	
Figure 46 - Run-Time: Create a Complaint by Citizen with JSON result	53	
Figure 47 - Run-Time: Create a Complaint by Citizen with XML result	53	
Figure 48 - Run-Time: Get List of Complaints with JSON result	54	
Figure 49 - Run-Time: Get List of Complaints with XML result.....	54	
Figure 50 - Run-Time: Reply to the Citizen's Complaint with JSON result	55	
Figure 51 - Run-Time: Reply to the Citizen's Complaint with XML result.....	55	
Figure 52-Sign-Up	Figure 53-Sign In	
Figure 54-Sign Up Citizen	Figure 55-Sign Up	
Citizen with data	57	
Figure 56-Selecting by NIC	Figure 57-Selecting by qualification	Figure 58-
Selecting by all records		58
Figure 59-Sign-In Admin		Figure 27-Admin home.....
Figure 60- Selecting All data		59
Email & Address, delete	Figure 28-Selecting	
Figure 61 - Swagger API Manager		60
Figure 62 - Execute Screenshot of the user_details table result	61	
Figure 63 - Execute Screenshot of the complaints table result	62	
Figure 64 - Execute Screenshot of the admin table result	62	
Figure 65 - The Full Backend API Part	63	
Figure 66- The Frontend Part (Citizen & Office: SignUp).....	65	
Figure 67 - The Frontend Part (Citizen & Office: SignIn)	66	
Figure 68 - The Frontend Part (Admin SignIn)	66	
Figure 69 - The Frontend Part (Admin Home Page with View & Delete Citizens' Information).....	67	
Figure 70 - LinkedIn certificate: 10707385	74	
Figure 71 - LinkedIn certificate: 10707241	75	
Figure 72 - LinkedIn certificate: 10707417	75	
Figure 73 - LinkedIn certificate: 10707129	76	
Figure 74 - LinkedIn certificate: 10707372	76	
Figure 75 - LinkedIn certificate: 10707291	77	

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 2022 and 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 requesting 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.

Sri Lanka Bureau of Foreign Employment (SLBFE) - REST API Web and Mobile Applications

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.

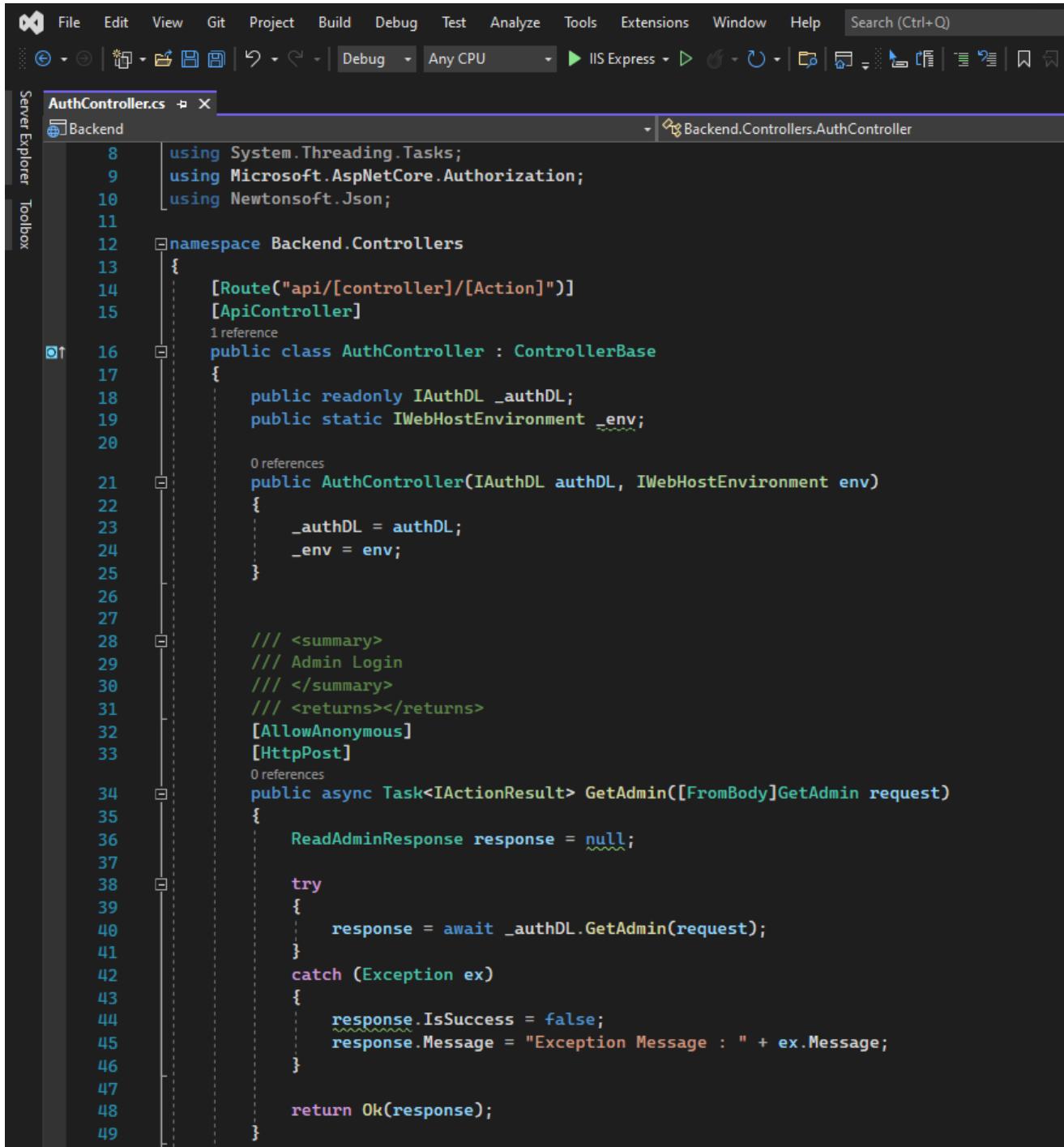
2.2. Screenshots

2.2.1. Code Screenshots - Backend

2.2.1.2. Web Application

Admin Sign-In

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



A screenshot of the Microsoft Visual Studio IDE interface. The title bar shows "File Edit View Git Project Build Debug Test Analyze Tools Extensions Window Help Search (Ctrl+Q)". The toolbar includes icons for file operations like Open, Save, and Print, along with build and run buttons. The status bar at the bottom shows "Backend.Controllers.AuthController". The main code editor window displays the "AuthController.cs" file under the "Backend" project. The code implements an API controller for user authentication. It includes imports for System.Threading.Tasks, Microsoft.AspNetCore.Authorization, and Newtonsoft.Json. The class is decorated with [Route("api/[controller]/[Action]")] and [ApiController]. It has a constructor taking IAuthDL and IWebHostEnvironment. The GetAdmin method is annotated with [AllowAnonymous] and [HttpPost], and it returns a Task<IActionResult>. The method logic involves trying to get an admin from the database and returning a response object if successful or an error message if an exception occurs.

```
8  using System.Threading.Tasks;
9  using Microsoft.AspNetCore.Authorization;
10 using Newtonsoft.Json;
11
12 namespace Backend.Controllers
13 {
14     [Route("api/[controller]/[Action]")]
15     [ApiController]
16     public class AuthController : ControllerBase
17     {
18         public readonly IAuthDL _authDL;
19         public static IWebHostEnvironment _env;
20
21         public AuthController(IAuthDL authDL, IWebHostEnvironment env)
22         {
23             _authDL = authDL;
24             _env = env;
25         }
26
27
28         /// <summary>
29         /// Admin Login
30         /// </summary>
31         /// <returns></returns>
32         [AllowAnonymous]
33         [HttpPost]
34         public async Task<IActionResult> GetAdmin([FromBody]GetAdmin request)
35         {
36             ReadAdminResponse response = null;
37
38             try
39             {
40                 response = await _authDL.GetAdmin(request);
41             }
42             catch (Exception ex)
43             {
44                 response.IsSuccess = false;
45                 response.Message = "Exception Message : " + ex.Message;
46             }
47
48             return Ok(response);
49         }
50     }
51 }
```

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

The screenshot shows the Microsoft Visual Studio IDE interface with the following details:

- File Bar:** File, Edit, View, Git, Project, Build, Debug, Test, Analyze, Tools, Extensions, Window, Help, Search (Ctrl+F).
- Toolbox:** Server Explorer, Toolbox.
- Project Explorer:** AuthDL.cs (Backend).
- Code Editor:** The main window displays the C# code for the `AuthDL.cs` file, specifically the `GetAdmin` method which handles the sign-in logic for administrators.
- Status Bar:** Debug, Any CPU, IS Express, Signin(SignInRequest request).

```
18  using ExcelDataReader;
19  using System.Data;
20  using System.Text.RegularExpressions;
21
22  namespace Backend.DataAccessLayer
23  {
24      [assembly: References]
25      public class AuthDL : IAuthDL
26      {
27          [assembly: References]
28          public readonly IConfiguration _configuration;
29          public readonly MySqlConnection _mySqlConnection;
30          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})?$/";
31
32          [assembly: References]
33          public readonly int ConnectionTimeOut = 180;
34
35          [assembly: References]
36          public async Task<ReadAdminResponse> GetAdmin(GetAdmin request)
37          {
38              ReadAdminResponse response = new ReadAdminResponse();
39              response.readAdmin = new List<GetAdmin>();
40
41              response.IsSuccess = true;
42              response.Message = "Successful";
43
44              try
45              {
46                  if (_mySqlConnection.State != System.Data.ConnectionState.Open)
47                  {
48                      await _mySqlConnection.OpenAsync();
49                  }
50
51                  //Check Null Or Empty
52                  if (string.IsNullOrEmpty(request.UserName) || string.IsNullOrEmpty(request.Password))
53                  {
54                      response.Message = "Username Or Password cannot Null or Empty";
55                  }
56
57                  //SQL Query
58                  string SqlQuery = @"SELECT *
59                                  FROM stbfe.admin
60                                  WHERE UserName=@UserName AND Password=@Password;";
61
62                  using (MySqlCommand sqlCommand = new MySqlCommand(SqlQuery, _mySqlConnection))
63                  {
64                      sqlCommand.CommandType = System.Data.CommandType.Text;
65                      sqlCommand.CommandTimeout = 180;
66                      sqlCommand.Parameters.AddWithValue("@UserName", request.UserName);
67                      sqlCommand.Parameters.AddWithValue("@Password", request.Password);
68
69                      using (DbDataReader dataReader = await sqlCommand.ExecuteReaderAsync())
70                      {
71                          if (dataReader.HasRows)
72                          {
73                              response.Message = "Login Successfull";
74                          }
75                          else
76                          {
77                              response.IsSuccess = false;
78                              response.Message = "Login Failed";
79                              return response;
80                          }
81                      }
82                  }
83              }
84              catch (Exception ex)
85              {
86                  response.IsSuccess = false;
87                  response.Message = ex.Message;
88              }
89              finally
90              {
91                  await _mySqlConnection.CloseAsync();
92                  await _mySqlConnection.DisposeAsync();
93              }
94          }
95
96          return response;
97      }
98  }
```

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 = "Username 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]
[HttpPost]
[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.

```
32
33     dependencies {
34
35         implementation 'androidx.appcompat:appcompat:1.4.1'
36         implementation 'com.google.android.material:material:1.6.0'
37         implementation 'androidx.constraintlayout:constraintlayout:2.1.3'
38         implementation 'androidx.lifecycle:lifecycle-livedata-ktx:2.3.1'
39         implementation 'androidx.lifecycle:lifecycle-viewmodel-ktx:2.3.1'
40         implementation 'androidx.navigation:navigation-fragment:2.3.5'
41         implementation 'androidx.navigation:navigation-ui:2.3.5'
42         testImplementation 'junit:junit:4.+'
43         androidTestImplementation 'androidx.test.ext:junit:1.1.3'
44         androidTestImplementation 'androidx.test.espresso:espresso-core:3.4.0'
45         implementation 'com.squareup.retrofit2:retrofit:2.9.0'
46         implementation 'com.squareup.retrofit2:converter-gson:2.9.0'
47     }
```

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();
        ModelCall.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(ModelApiManager modelApiManager)
{
    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

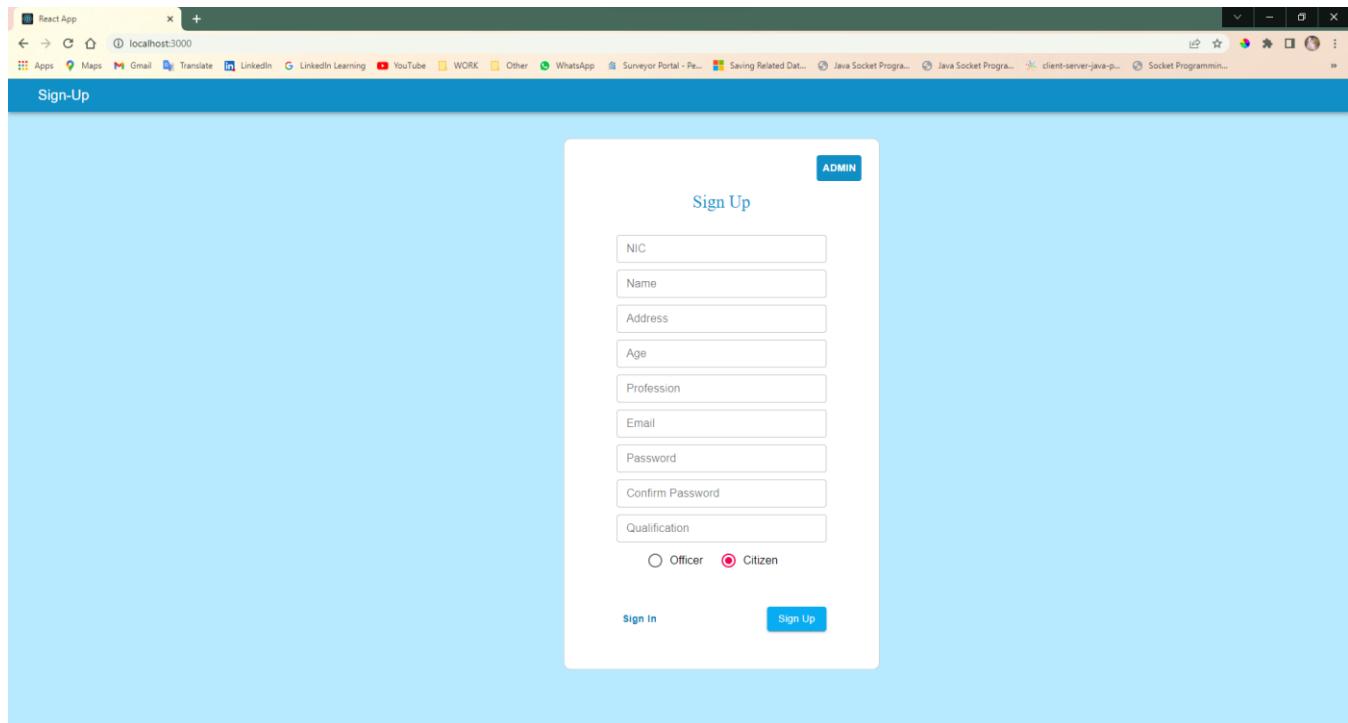


Figure 24 - Run-Time: Citizen & Officer Sign-Up User-Interface

The screenshot shows a REST API testing interface. At the top, a green bar indicates a **POST** method to the endpoint **/api/Auth/SignUp/Create**. Below this, under the **Parameters** section, it says "No parameters". In the **Request body** section, there is a dropdown set to **application/json**, and a JSON payload is displayed:

```
{
  "nic": "954376653V",
  "name": "Kasun",
  "address": "No.55, Colombo",
  "age": 27,
  "profession": "Teacher",
  "email": "kasun@gmail.com",
  "password": "1234",
  "configPassword": "1234",
  "affiliation": "Citizen",
  "qualification": "Teaching"
}
```

Below the request body, there are two buttons: **Execute** (blue) and **Clear** (grey). Under the **Responses** section, there is a **Curl** command and a **Request URL** (both in black boxes). Under the **Server response** section, there are tabs for **Code** and **Details**. The **Code** tab shows a status of **200** and a response body with a success message:

```
{"isSuccess": true, "message": "Successful"}
```

The **Details** tab shows response headers:

```
access-control-allow-origin: *
content-type: application/json; charset=utf-8
date: Tue 10 May 2022 20:33:48 GMT
```

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

The screenshot shows a REST API testing interface. At the top, a red bar indicates a **POST** method to the endpoint **https://localhost:44381/api/Auth/SignUp/Create**. Below this, under the **Body** tab, there are tabs for **Params**, **Authorization**, **Headers (8)**, **Body** (green), **Pre-request Script**, **Tests**, and **Settings**. The **Body** tab is selected, showing the XML payload:

```

1 <ns>
2   <ns>nic: "954376653V",
3   <ns>name: "Kasun",
4   <ns>address: "No.55, Colombo",
5   <ns>age: 27,
6   <ns>profession: "Teacher",
7   <ns>email: "kasun@gmail.com",
8   <ns>password: "1234",
9   <ns>configPassword: "1234",
10  <ns>affiliation: "Officer",
11  <ns>qualification: "Teaching"
12 </ns>

```

Below the body, there are tabs for **Body**, **Cookies**, **Headers (5)**, and **Test Results**. The **Test Results** tab is selected, showing a status of **200 OK**, time of **38 ms**, size of **223 B**, and a **Save Response** button. The XML response is displayed:

```

1 {"isSuccess":true,"message":"Successful"}

```

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

Citizen & Officer - Sign-In Page

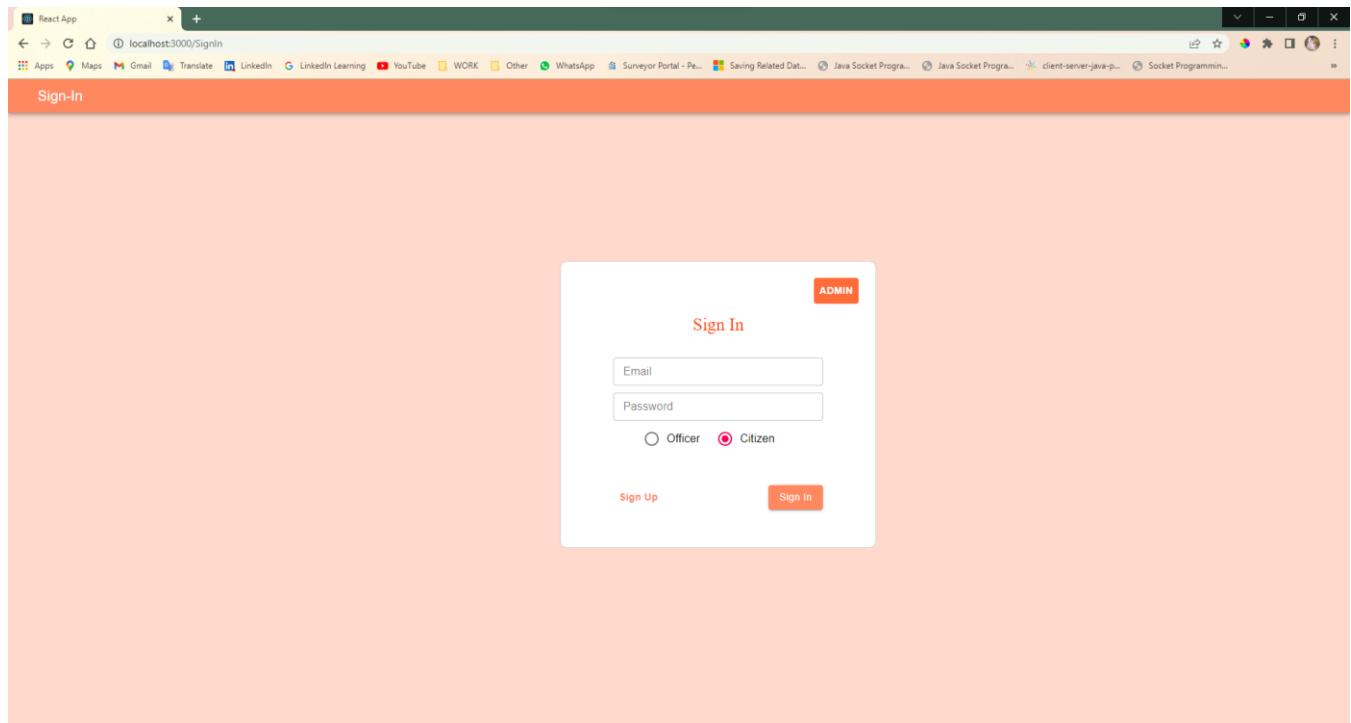


Figure 27 - Run-Time: Citizen & Officer Sign-In Use-Interface

A screenshot of a POST request to `/api/Auth/SignIn`. The request body is JSON:

```
{ "email": "neranji@gmail.com", "password": "1234", "affiliation": "Citizen" }
```

The response shows a successful status (200) with the following JSON body:

```
{ "isSuccess": true, "message": "Login Successfull" }
```

Response headers include:

```
access-control-allow-origin: *
content-type: application/json; charset=utf-8
date: Tue 10 May 2022 20:39:49 GMT
server: Microsoft-IIS/10.0
```

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

The screenshot shows the Postman application interface. A POST request is being made to <https://localhost:44381/api/Auth/SignIn>. The request body is set to JSON, containing the following data:

```
1 <?
2   "email": "neranji@gmail.com",
3   "password": "1234",
4   "affiliation": "Citizen"
5 </?>
```

The response status is 200 OK, with a time of 27 ms and a size of 230 B. The response body is XML, indicating success:

```
1 {"isSuccess":true,"message":"Login Successfull"}
```

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

Admin Sign-In Page

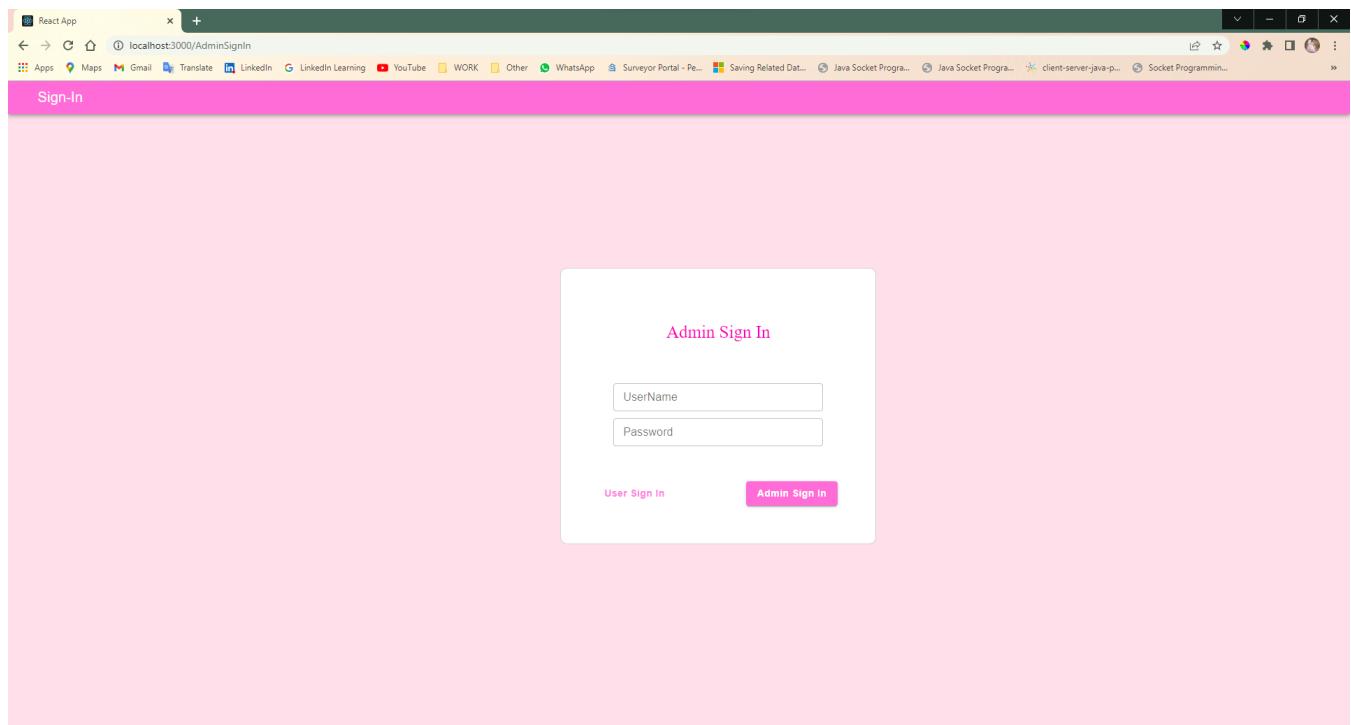


Figure 30 - Run-Time: Admin Sign-In User-Interface

A screenshot of the Postman application interface. The top bar shows 'POST /api/Auth/GetAdmin'. The 'Parameters' section is empty. The 'Request body' section is set to 'application/json' and contains the following JSON payload:

```
{  
  "userName": "AdminSLBFE",  
  "password": "Admin123"  
}
```

The 'Execute' button is highlighted in blue at the bottom left. The 'Responses' section is collapsed. The 'Code' tab is selected, showing the status code '200' and the response body:

```
{  
  "isSuccess": true,  
  "message": "Login Successfull",  
  "readAdmin": []  
}
```

The 'Details' tab is also visible. The 'Server response' section is collapsed.

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

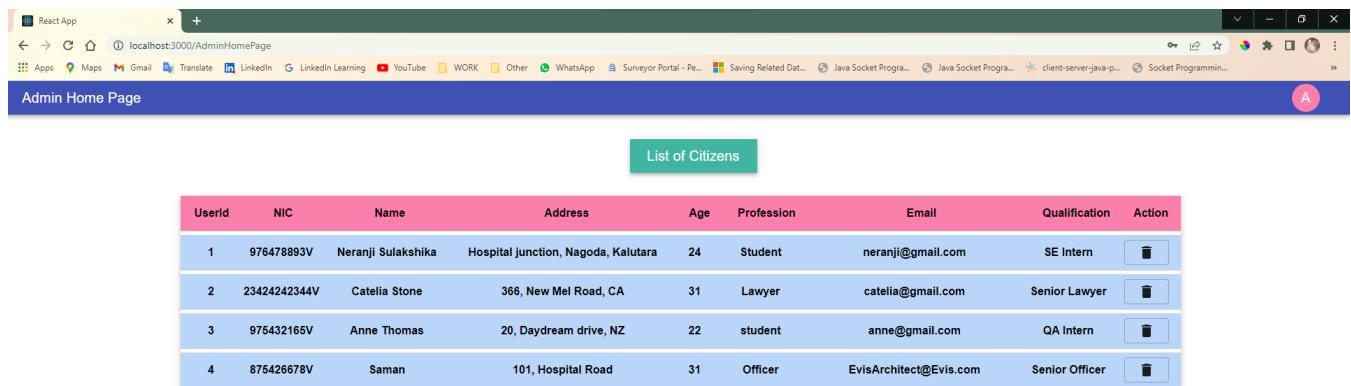
The screenshot shows a REST client interface with the following details:

- URL:** https://localhost:44381/api/Auth/ReadCitizenInformation
- Method:** GET
- Headers:** (6) - This request does not have a body.
- Body:** (Pretty, Raw, Preview, Visualize, XML) - Displays the XML response.
- Response Status:** 200 OK, Time: 36 ms, Size: 1.11 KB
- XML Response Content:**

```
1  {"isSuccess":true,"message":"Successful","readCitizenInformation":[{"userId":1,"nic":"976478893V","name":"Neeranji Sulakshika","address":"Hospital junction, Nagoda, Kalutara","age":24,"profession":"Student","email":"neeranj@gmail.com","qualification":"SE Intern"}, {"userId":2,"nic":"23424242344V","name":"Catelia Stone","address": "366, New Mel Road, CA","age":31,"profession":"Lawyer","email":"catelia@gmail.com","qualification":"Senior Lawyer"}, {"userId":3,"nic":"975432168V","name":"Anne Thomas","address":"20, Daydream drive, NZ","age":22,"profession":"student","email":"anne@gmail.com","qualification":"QA Intern"}, {"userId":4,"nic":"875426678V","name":"Saman","address": "101, Hospital Road","age":31,"profession":"Officer","email":"EvisArchitect@Evis.com","qualification":"Senior Officer"}, {"userId":5,"nic":"954376653V","name":"Kasun","address": "No.55, Colombo","age":27,"profession":"Teacher","email":"kasun@gmail.com","qualification":"Teaching"}]}
```

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

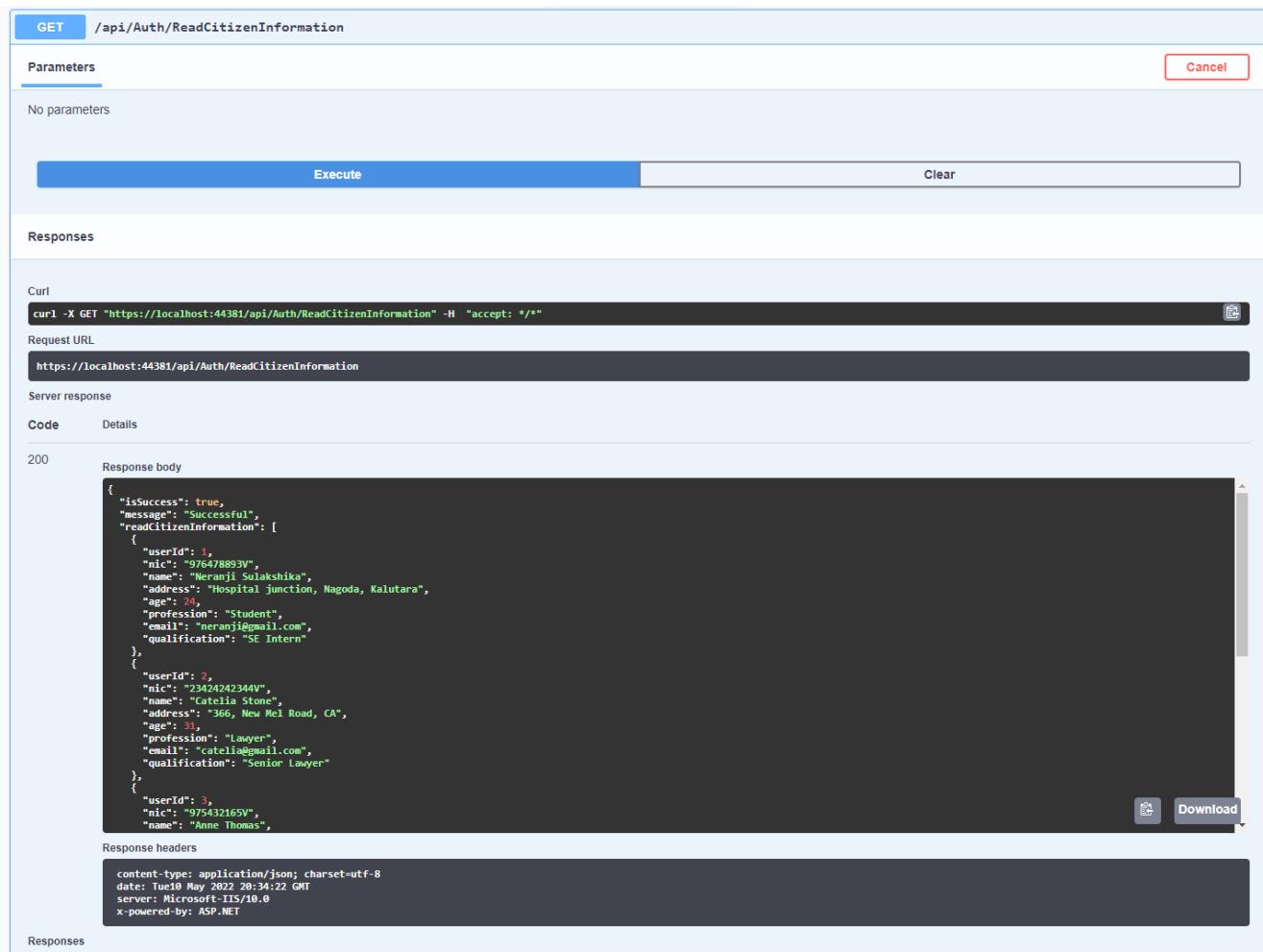
Admin Home Page



The screenshot shows a web browser window titled "React App" with the URL "localhost:3000/AdminHomePage". The page has a blue header bar with the text "Admin Home Page". Below the header is a green button labeled "List of Citizens". The main content is a table with the following data:

UserId	NIC	Name	Address	Age	Profession	Email	Qualification	Action
1	976478893V	Neranji Sulakshika	Hospital junction, Nagoda, Kalutara	24	Student	neranji@gmail.com	SE Intern	
2	23424242344V	Catelia Stone	366, New Mel Road, CA	31	Lawyer	catelia@gmail.com	Senior Lawyer	
3	975432165V	Anne Thomas	20, Daydream drive, NZ	22	student	anne@gmail.com	QA Intern	
4	875426678V	Saman	101, Hospital Road	31	Officer	EvisArchitect@Evis.com	Senior Officer	

Figure 33 - Admin Home User-Interface



The screenshot shows a Swagger UI interface for a "GET /api/Auth/ReadCitizenInformation" endpoint. The "Parameters" section indicates "No parameters". The "Responses" section shows a "Curl" example and a "Server response" in JSON format.

Responses

Server response

```

Curl
curl -X GET "https://localhost:44381/api/Auth/ReadCitizenInformation" -H "accept: */*"
Request URL
https://localhost:44381/api/Auth/ReadCitizenInformation
Server response
Code Details
200 Response body
{
  "isSuccess": true,
  "message": "Successful",
  "readCitizenInformation": [
    {
      "userId": 1,
      "nic": "976478893V",
      "name": "Neranji Sulakshika",
      "address": "Hospital junction, Nagoda, Kalutara",
      "age": 24,
      "profession": "Student",
      "email": "neranji@gmail.com",
      "qualification": "SE Intern"
    },
    {
      "userId": 2,
      "nic": "23424242344V",
      "name": "Catelia Stone",
      "address": "366, New Mel Road, CA",
      "age": 31,
      "profession": "Lawyer",
      "email": "catelia@gmail.com",
      "qualification": "Senior Lawyer"
    },
    {
      "userId": 3,
      "nic": "975432165V",
      "name": "Anne Thomas"
    }
  ]
}
  
```

Response headers

```

content-type: application/json; charset=utf-8
date: Tue 10 May 2022 20:34:22 GMT
server: Microsoft-IIS/10.0
x-powered-by: ASP.NET
  
```

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

The screenshot shows the Postman application interface. At the top, there's a header with 'Overview' and a green 'GET https://localhost:44381...' button. To the right, it says 'No Environment'. Below the header, the URL 'https://localhost:44381/api/Auth/ReadCitizenInformation' is entered. On the left, there are tabs for 'Params', 'Authorization', 'Headers (6)', 'Body', 'Pre-request Script', 'Tests', and 'Settings'. The 'Body' tab is selected. Under 'Body', there are tabs for 'Pretty', 'Raw', 'Preview', 'Visualize', and 'XML'. The 'XML' tab is selected. The response body is displayed as XML:

```
1  {"isSuccess":true,"message":"Successful","readCitizenInformation":[{"userId":1,"nic":"976478893V","name":"Neranji Sulakshika","address":"Hospital junction, Nagoda, Kalutara","age":24,"profession":"Student","email":"neranji@gmail.com","qualification":"SE Intern"}, {"userId":2,"nic":"23424242344V","name":"Catelia Stone","address": "366, New Mel Road, CA","age":31,"profession":"Lawyer","email":"catelia@gmail.com","qualification":"Senior Lawyer"}, {"userId":3,"nic":"975432165V","name":"Anne Thomas","address": "20, Daydream drive, NZ","age":22,"profession":"student","email":"anne@gmail.com","qualification":"QA Intern"}, {"userId":4,"nic":"876426678V","name":"Saman","address": "101, Hospital Road","age":31,"profession":"Officer","email":"EvisArchitect@evis.com","qualification":"Senior Officer"}]}
```

To the right of the XML, status information is shown: Status: 200 OK, Time: 42 ms, Size: 977 B, with a 'Save Response' button.

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

Delete Citizen Account

The screenshot shows a REST API tool interface. At the top, there is a red header bar with the URL `DELETE /api/Auth/DeleteCitizen/Delete/{UserId}`. Below this is a section titled "Parameters" with a table:

Name	Description
request object (header)	<pre>{ "userId": 5 }</pre>

Below the table is a field labeled "UserId * required string (path)" with the value "5". To the right of this field is a "Cancel" button and a small circular icon with a question mark.

At the bottom of the interface are two buttons: "Execute" (in blue) and "Clear".

Under the "Responses" section, there is a "Curl" block containing the command:

```
curl -X DELETE "https://localhost:44381/api/Auth/DeleteCitizen/Delete/5" -H "accept: */*" -H "request: userId,5"
```

Below the curl command is a "Request URL" field with the value "https://localhost:44381/api/Auth/DeleteCitizen/Delete/5".

The "Server response" section shows a "Code" tab selected, displaying the status code "200". Under "Details", the "Response body" is shown as:

```
{
  "isSuccess": true,
  "message": "Successful"
}
```

Next to the response body are "Copy" and "Download" buttons. Below the response body is a "Response headers" block containing:
access-control-allow-origin: *
content-type: application/json; charset=utf-8
date: Wed 11 May 2022 06:39:43 GMT
server: Microsoft-IIS/10.0
x-powered-by: ASP.NET

At the bottom of the "Responses" section is a "Responses" link.

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

The screenshot shows a REST API tool interface. At the top, there is a header bar with the URL `DEL https://localhost:44381`. Below this is a search bar with the value "https://localhost:44381/api/Auth/DeleteCitizen/Delete/8".

The main area shows a "DELETE" button and the URL "https://localhost:44381/api/Auth/DeleteCitizen/Delete/8". Below these are tabs for "Params", "Authorization", "Headers (6)", "Body", "Pre-request Script", "Tests", and "Settings". The "Body" tab is selected, showing the following content:

None

This request does not have a body.

At the bottom right, there is a status bar showing "Status: 200 OK Time: 34 ms Size: 223 B Save Response".

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

Find Citizen Information by NIC

GET /api/Auth/CitizenInformationByNIC/Get/{NIC}

Parameters

Name	Description
request object (header)	{ "nic": "976478893V" }

NIC * required
string (path) 976478893V

Execute Clear

Responses

Curl

```
curl -X GET "https://localhost:44381/api/Auth/CitizenInformationByNIC/Get/976478893V" -H "accept: /*" -H "request: nic,976478893V"
```

Request URL

https://localhost:44381/api/Auth/CitizenInformationByNIC/Get/976478893V

Server response

Code Details

200 Response body

```
{
  "isSuccess": true,
  "message": "Successful",
  "getCitizenInformationByNIC": {
    "name": "Neranji Sulakshika",
    "address": "Hospital junction, Nagoda, Kalutara",
    "age": 24,
    "profession": "Student",
    "email": "neranji@gmail.com",
    "qualification": "SE Intern"
  }
}
```

Download

Response headers

```
content-type: application/json; charset=utf-8
date: Tue 10 May 2022 20:42:20 GMT
server: Microsoft-IIS/10.0
```

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

Overview GET https://localhost:44381/ No Environment

https://localhost:44381/api/Auth/CitizenInformationByNIC/Get/976478893V

GET https://localhost:44381/api/Auth/CitizenInformationByNIC/Get/976478893V Send

Params Authorization Headers (8) Body Pre-request Script Tests Settings Cookies Beautify

none form-data x-www-form-urlencoded raw binary GraphQL JSON

```

1   {
2     "nic": "976478893V"
3   }

```

Body Cookies Headers (5) Test Results Status: 200 OK Time: 16 ms Size: 418 B Save Response

Pretty Raw Preview Visualize XML

```

1   {"isSuccess":true,"message":"Successful","getCitizenInformationByNIC":{"name":"Neranji Sulakshika","address":"Hospital junction, Nagoda, Kalutara","age":24,"profession":"Student","email":"neranji@gmail.com","qualification":"SE Intern"}}

```

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

Find Citizen Information by Qualification

GET /api/Auth/CitizenInformationByQualification/Get/{Qualification}

Parameters

Name	Description
request object (header)	{ "qualification": "SE Intern" }

Qualification * required
string (path)
SE Intern

Execute Clear

Responses

Curl

```
curl -X GET "https://localhost:44381/api/Auth/CitizenInformationByQualification/Get/SEIntern" -H "accept: */*" -H "request: qualification,SE Intern"
```

Request URL

<https://localhost:44381/api/Auth/citizeninformationbyqualification/get/seintern>

Server response

Code Details

200 Response body

```
{
  "isSuccess": true,
  "message": "Successful",
  "getCitizenInformationByQualification": [
    {
      "nic": "976478893V",
      "name": "Neranji Sulakshika",
      "address": "Hospital junction, Nagoda, Kalutara",
      "age": 24,
      "profession": "Student",
      "email": "neranji@gmail.com"
    }
  ]
}
```

Response headers

```
content-type: application/json; charset=utf-8
```

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

Overview GET https://localhost:44381/ + ⚡

No Environment

https://localhost:44381/api/Auth/CitizenInformationByQualification/Get/SE Intern

GET https://localhost:44381/api/Auth/CitizenInformationByQualification/Get/SE Intern

Send

Params Authorization Headers (8) Body Pre-request Script Tests Settings Cookies Beautify

none form-data x-www-form-urlencoded raw binary GraphQL JSON

```
1 {
2   "qualification": "SE Intern"
3 }
```

Body Cookies Headers (5) Test Results

Pretty Raw Preview Visualize XML

Status: 200 OK Time: 15 ms Size: 421 B Save Response

```
1 { "isSuccess":true,"message":"Successful","getCitizenInformationByQualification": [{"nic": "976478893V", "name": "Neranji Sulakshika", "address": "Hospital junction, Nagoda, Kalutara", "age": 24, "profession": "Student", "email": "neranji@gmail.com"}]}
```

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

Update Citizen Qualification by NIC

PUT /api/Auth/UpdateCitizenInformation/UpdateQualification/{NIC}

Parameters

Name	Description
NIC * required	23424242344V (path)

Request body

```
{
  "nic": "23424242344V",
  "qualification": "Lawyer"
}
```

Execute **Clear**

Responses

Curl

```
curl -X PUT "https://localhost:44381/api/Auth/UpdateCitizenInformation/UpdateQualification/23424242344V" -H "accept: */*" -H "Content-Type: application/json" -d "{\"nic\":\"23424242344V\",\"qualification\":\"Lawyer\"}"
```

Request URL

<https://localhost:44381/api/Auth/UpdateCitizenInformation/UpdateQualification/23424242344V>

Server response

Code **Details**

200 **Response body**

```
{
  "isSuccess": true,
  "message": "Successful"
}
```

Download

Response headers

```
access-control-allow-origin: *
content-type: application/json; charset=utf-8
date: Tue, 10 May 2022 20:50:08 GMT
server: Microsoft-IIS/10.0
x-powered-by: ASP.NET
```

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

Overview **PUT https://localhost:44381** **+** ******* **No Environment** **✓**

<https://localhost:44381/api/Auth/UpdateCitizenInformation/UpdateQualification/23424242344V>

PUT <https://localhost:44381/api/Auth/UpdateCitizenInformation/UpdateQualification/23424242344V> **Send**

Params **Authorization** **Headers (8)** **Body** **Pre-request Script** **Tests** **Settings** **Cookies**

none form-data x-www-form-urlencoded raw binary GraphQL **JSON** **Beautify**

```

1 <pre>
2   "nic": "23424242344V",
3   "qualification": "Senior Lawyer"
4 </pre>

```

Body **Cookies** **Headers (5)** **Test Results** **Status: 200 OK** **Time: 16 ms** **Size: 223 B** **Save Response**

Pretty **Raw** **Preview** **Visualize** **XML** **JSON**

```

1 {"isSuccess":true,"message":"Successful"}

```

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

Upload Citizen Documents

POST /api/Auth/UploadCitizenDocuments

Parameters

No parameters

Request body

file
string(\$binary) 83.png

Send empty value

Execute **Clear**

Responses

Curl
`curl -X POST "https://localhost:44381/api/Auth/UploadCitizenDocuments" -H "accept: text/plain" -H "Content-type: multipart/form-data" -F "file=@83.png;type=image/png"`

Request URL
<https://localhost:44381/api/Auth/UploadCitizenDocuments>

Server response

Code	Details
200	Response body <pre>File_777.png</pre> Response headers <pre>access-control-allow-origin: * content-encoding: gzip content-type: text/plain; charset=utf-8 date: Wed 11 May 2022 06:34:39 GMT server: Microsoft-IIS/10.0 vary: Accept-Encoding x-powered-by: ASP.NET</pre>

Responses

Code	Description	Links
200	Success	No links

Media type

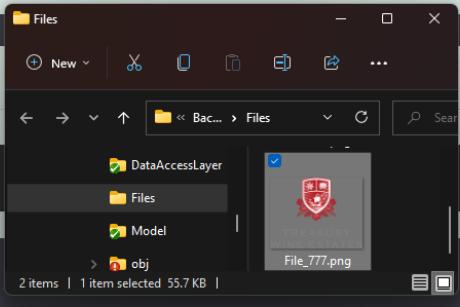


Figure 44 - Run-Time: Upload Citizen Documents with JSON result

Overview **POST https://localhost:44381/api/Auth/UploadCitizenDocuments** + **...** **No Environment** **Save** **Send** **Cookies**

https://localhost:44381/api/Auth/UploadCitizenDocuments

POST <https://localhost:44381/api/Auth/UploadCitizenDocuments>

Params **Authorization** **Headers (8)** **Body** **Pre-request Script** **Tests** **Settings** **Cookies**

none form-data x-www-form-urlencoded raw binary GraphQL

KEY	VALUE	DESCRIPTION	...	Bulk Edit
<input checked="" type="checkbox"/> file	signature.png			

Body **Cookies** **Headers (7)** **Test Results** **Pretty** **Raw** **Preview** **Visualize** **XML** **...**

Status: 200 OK **Time:** 123 ms **Size:** 235 B **Save Response**

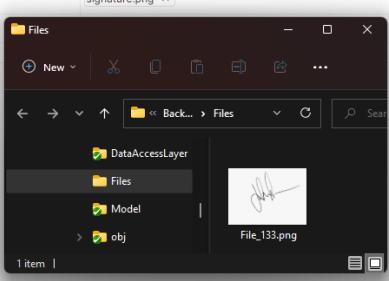


Figure 45 - Run-Time: Upload Citizen Documents with XML result

Create a Complaint by Citizen

The screenshot shows a REST API testing interface. At the top, a green bar indicates a successful POST request to `/api/Auth/CreateComplaint`. Below this, the 'Parameters' section shows 'No parameters'. The 'Request body' section contains the following JSON:

```
{ "complaint": "Very bad" }
```

The 'Responses' section shows a successful 200 response. The 'Code' tab is selected, displaying the response body:

```
{"isSuccess": true, "message": "Successful"}
```

The 'Details' tab shows the response headers:

```
access-control-allow-origin: *  
content-type: application/json; charset=utf-8  
date: Wed 11 May 2022 06:45:05 GMT  
server: Microsoft-IIS/10.0  
x-powered-by: ASP.NET
```

At the bottom, there are 'Responses' and 'Links' sections.

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

The screenshot shows a REST API testing interface. At the top, a green bar indicates a successful POST request to `https://localhost:44381/api/Auth/CreateComplaint`. Below this, the 'Body' tab is selected, showing the XML body sent to the server:

```
<complaint>Bad</complaint>
```

The 'Test Results' section shows a successful 200 OK response with a status of 200 OK, time 45 ms, and size 223 B. The response body is identical to the one in Figure 46:

```
{"isSuccess": true, "message": "Successful"}
```

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

Get List of Complaints

The screenshot shows a REST API tool interface. At the top, a blue bar indicates the method as 'GET' and the endpoint as '/api/Auth/ReadComplaints'. Below this, there's a 'Parameters' section with a 'Cancel' button. The 'Responses' section is expanded, showing a 'Curl' command to run the request, the 'Request URL' (https://localhost:44381/api/Auth/ReadComplaints), and the 'Server response'.

Server response:

Code Details

200 Response body

```
{ "isSuccess": true, "message": "Successful", "readComplaints": [ { "complaintId": 1, "complaint": "Very bad", "reply": "" }, { "complaintId": 2, "complaint": "bad", "reply": "" } ] }
```

Download

Response headers

```
content-type: application/json; charset=utf-8
date: Wed, 11 May 2022 06:48:06 GMT
server: Microsoft-IIS/10.0
x-powered-by: ASP.NET
```

Responses

Code Description Links

200 Success No links

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

The screenshot shows a REST API tool interface. At the top, it says 'Overview' and 'GET https://localhost:44381/api/Auth/ReadComplaints'. Below this, there's a 'Params' section with a 'Send' button. The 'Body' tab is selected, showing the XML response.

Body Cookies Headers (5) Test Results

Pretty Raw Preview Visualize XML

```
1  {"isSuccess":true,"message":"Successful","readComplaints":[{"complaintId":1,"complaint":"Very bad","reply":""}, {"complaintId":2,"complaint":"Bad","reply":""}]} 
```

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

Reply to the Citizen's Complaint

The screenshot shows a REST API testing interface with the following details:

- Method:** PUT
- Endpoint:** /api/Auth/UpdateComplaintInformation/Update/{ComplaintId}
- Parameters:**

Name	Description
ComplaintId * required	string (path)
1	
- Request body:**

```
{
  "complaintId": 1,
  "reply": "Thank you"
}
```
- Content-Type:** application/json
- Responses:**
 - Curl:** curl -X PUT "https://localhost:44381/api/Auth/UpdateComplaintInformation/Update/1" -H "accept: /*" -H "Content-Type: application/json" -d "{\"complaintId\":1,\"reply\":\"Thank you\"}"
 - Request URL:** https://localhost:44381/api/Auth/UpdateComplaintInformation/Update/1
 - Server response:**

Code	Details
200	Response body: <pre>{ "isSuccess": true, "message": "Successful" }</pre> Response headers: <pre>access-control-allow-origin: * content-type: application/json; charset=utf-8 date: Wed 11 May 2022 06:51:18 GMT server: Microsoft-IIS/10.0 x-powered-by: ASP.NET</pre>

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

The screenshot shows a REST API testing interface with the following details:

- Method:** PUT
- Endpoint:** https://localhost:44381/api/Auth/UpdateComplaintInformation/Update/2
- Params:** none
- Body:**

```

1 <root>
2   <complaintId> 2</complaintId>
3   <reply> Thanks</reply>
4 </root>

```
- Response:**
 - Status:** 200 OK
 - Time:** 32 ms
 - Size:** 223 B
 - Save Response:** checked

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

The screenshot shows the 'Sign Up' screen for a citizen. At the top, there is a header with the SLBFE logo and a circular progress bar. Below the header, the title 'Sign Up' is displayed. The form consists of several input fields: NIC (915876348V), Name (Dias Almeda), Age (30), Address (102/56 Galle Rd, Kollupitiya), Profession (Business Man), Email (diasalmeda95@gmail.com), Password (*****), and Confirm Password (*****). Below these fields are two radio buttons: 'Citizen' (selected) and 'Officer'. A green 'Sign Up' button is located at the bottom right. At the very bottom, there is a link 'Already a User? Sign In'.

Dashboard / Home page => Citizen

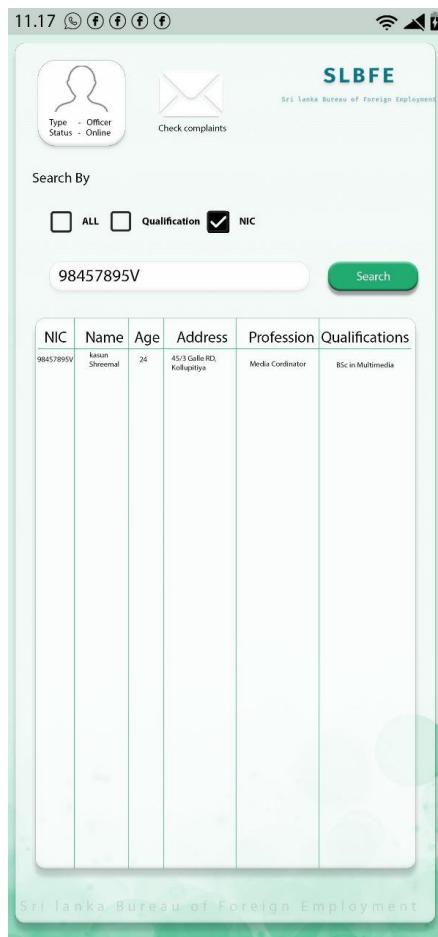
Figure 55-Sign Up Citizen with data

The screenshot shows the citizen's profile page. At the top, it displays 'Mobitel' and the time '10:58 PM'. The SLBFE logo and the text 'Sri Lanka Bureau of Foreign Employment' are at the top left. On the right, there is a user icon with the text 'Type - Citizen' and 'Status - Online'. Below this is a 'Profile' section with the following details:
Name - Dias Almeda
Address - 102/56 Galle Rd, Pannipitiya.
Age - 27
Profession - Business Man
Email - diasalmeda95@gmail.com
Below the profile, there is a green 'Edit Profile' button. Further down, there are sections for uploading documents: 'Cv' (Upload), 'Passport' (Upload), 'Qualifications' (Upload), and 'Birth Certificate' (Upload). A note states: '*Note - Please Upload following documents.' and 'Cv', 'Passport', 'Qualifications', and 'Birth Certificate' each have an 'Upload' button. At the bottom, there is a 'Complain ? Upload' section with a note: '*Note- You must fill a complain form first, you can download it in your profile'. The footer of the screen also says 'Sri Lanka Bureau of Foreign Employment'.

- When a citizen signs up 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 log in 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



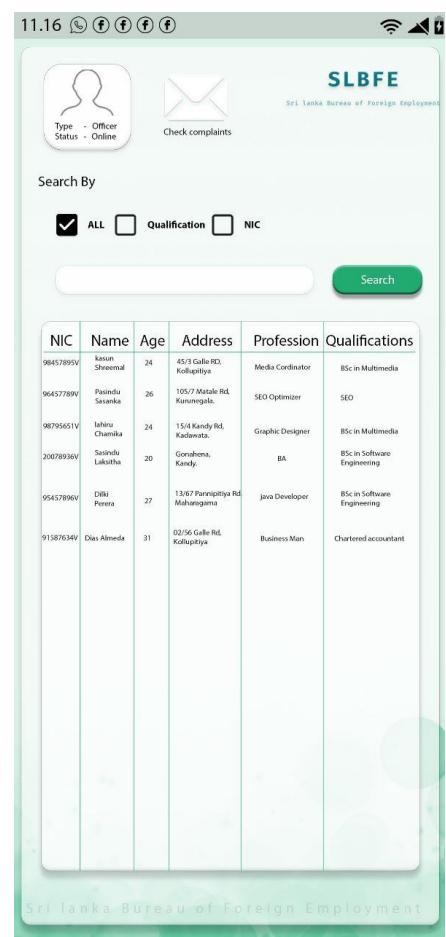
Selecting by Qualification

Figure 56-Selecting by NIC



Selection of all records

Figure 57-Selecting by qualification



- 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

Home Page => Home Page / Dashboard

Figure 59-Sign-In Admin



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.

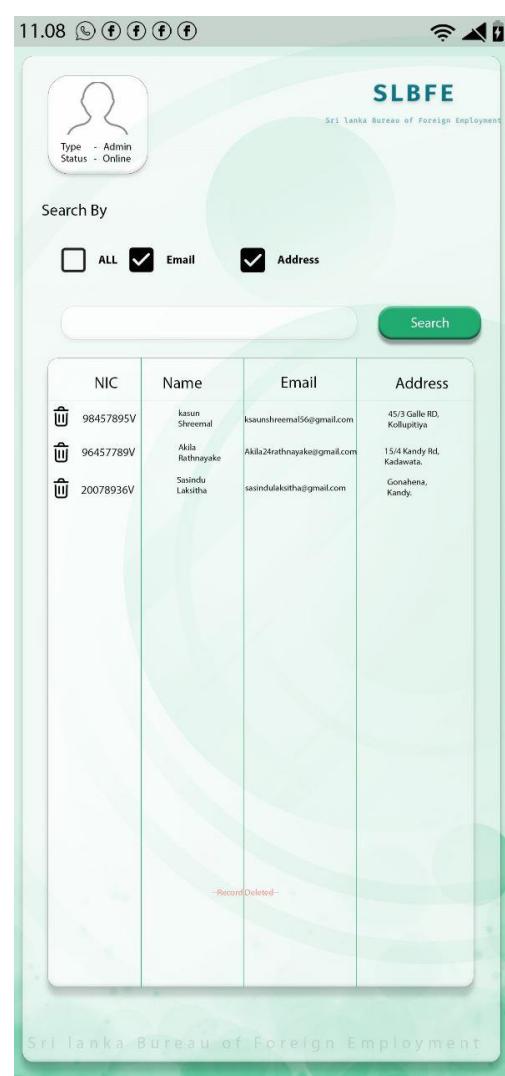
Selecting all records

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.

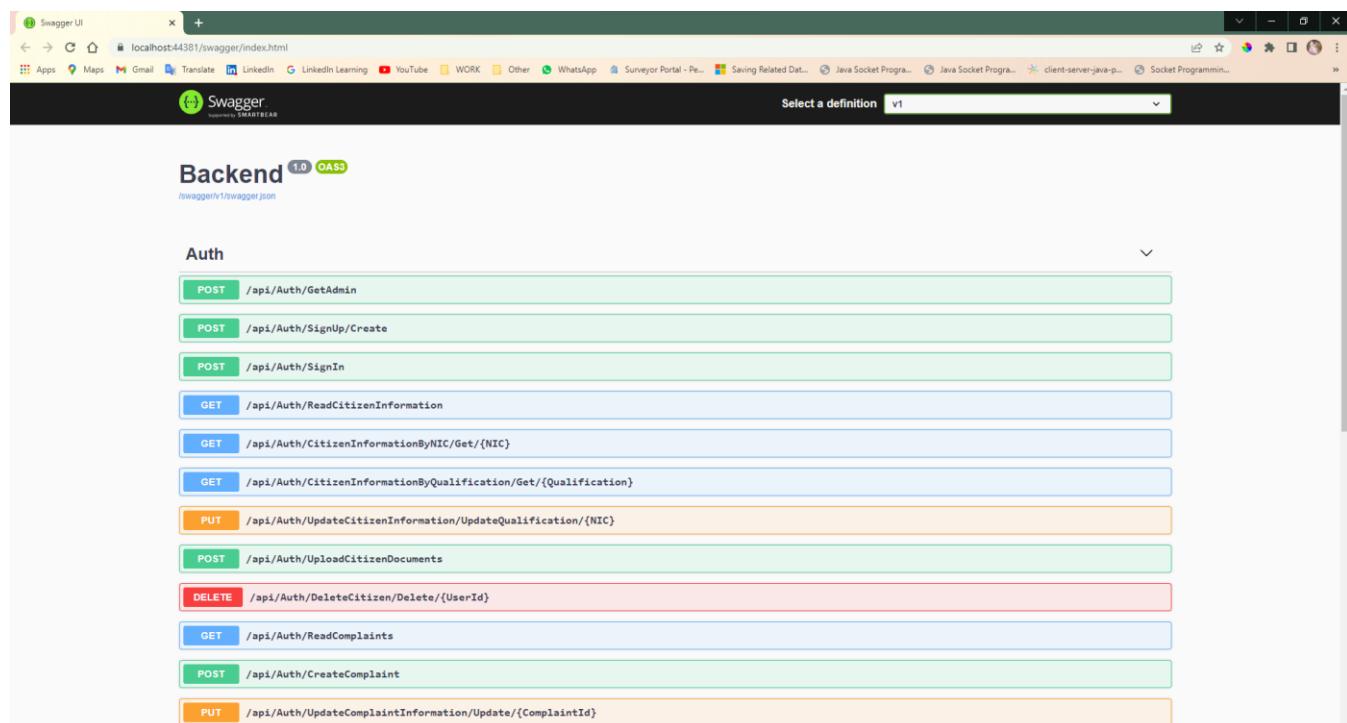
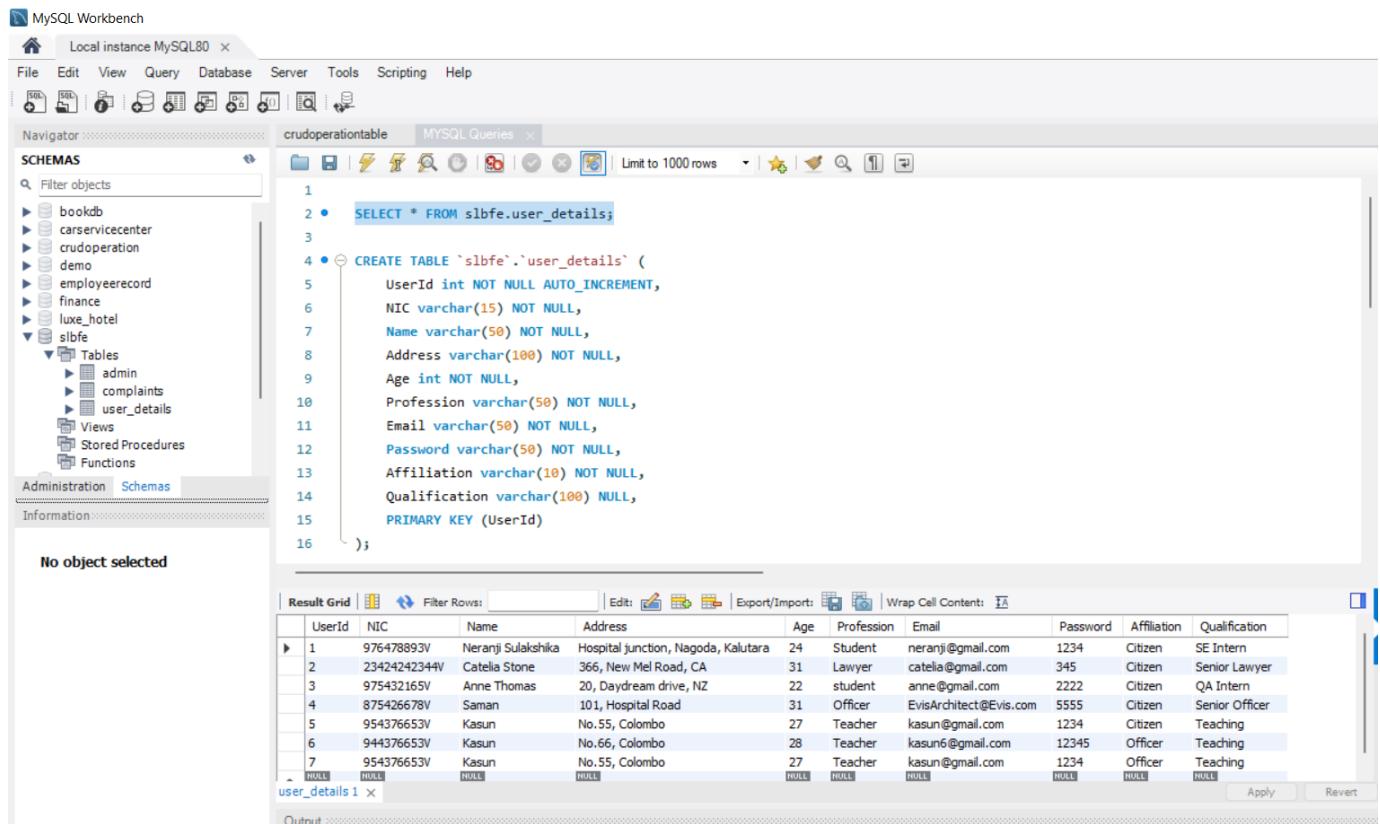


Figure 61 - Swagger API Manager

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



This screenshot shows the MySQL Workbench interface with the 'MySQL Queries' tab active. The code pane displays the creation of the 'user_details' table:

```

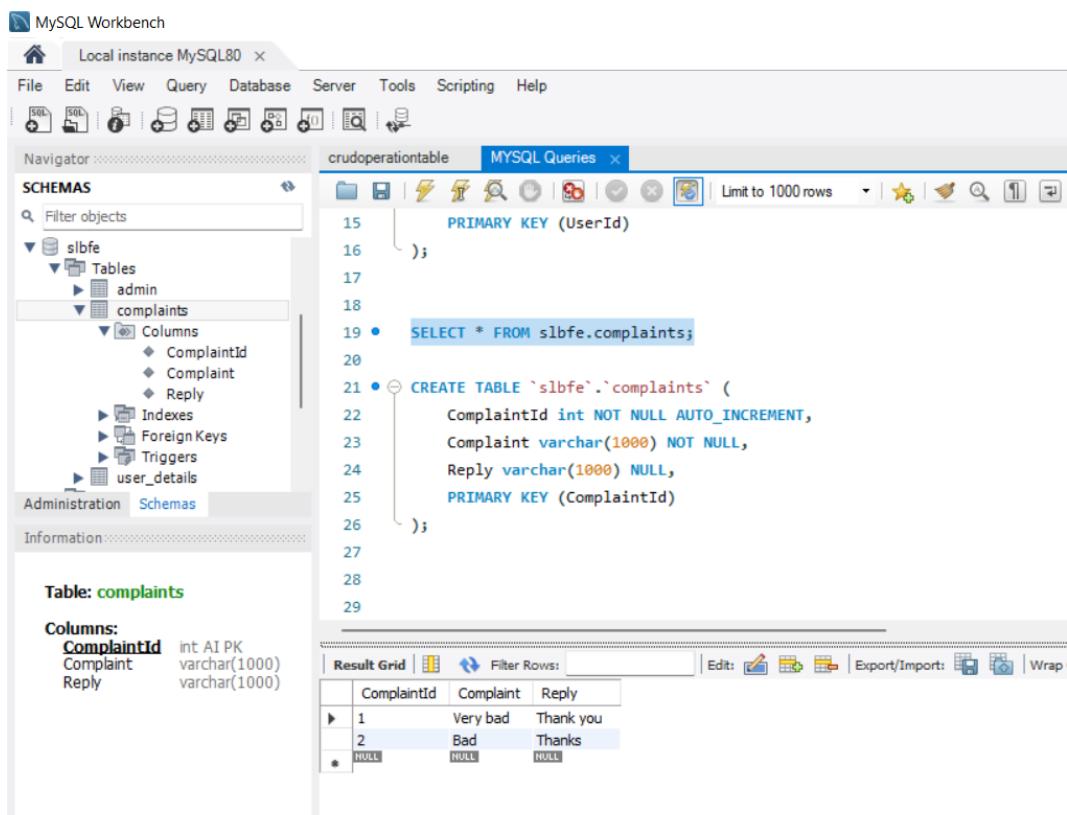
1
2 • SELECT * FROM slbfe.user_details;
3
4 • CREATE TABLE `slbfe`.`user_details` (
5     UserId int NOT NULL AUTO_INCREMENT,
6     NIC varchar(15) NOT NULL,
7     Name varchar(50) NOT NULL,
8     Address varchar(100) NOT NULL,
9     Age int NOT NULL,
10    Profession varchar(50) NOT NULL,
11    Email varchar(50) NOT NULL,
12    Password varchar(50) NOT NULL,
13    Affiliation varchar(10) NOT NULL,
14    Qualification varchar(100) NULL,
15    PRIMARY KEY (UserId)
16 );

```

The results grid below shows the data for the 'user_details' table:

User Id	NIC	Name	Address	Age	Profession	Email	Password	Affiliation	Qualification
1	976478893V	Neranj Sulakshika	Hospital junction, Nagoda, Kalutara	24	Student	neranj@gmail.com	1234	Citizen	SE Intern
2	23424242344V	Catela Stone	366, New Mel Road, CA	31	Lawyer	catela@gmail.com	345	Citizen	Senior Lawyer
3	975432165V	Anne Thomas	20, Daydream drive, NZ	22	student	anne@gmail.com	2222	Citizen	QA Intern
4	875426678V	Saman	101, Hospital Road	31	Officer	EvisArchitect@Evis.com	5555	Citizen	Senior Officer
5	954376653V	Kasun	No.55, Colombo	27	Teacher	kasun@gmail.com	1234	Citizen	Teaching
6	944376653V	Kasun	No.66, Colombo	28	Teacher	kasun6@gmail.com	12345	Officer	Teaching
7	954376653V	Kasun	No.55, Colombo	27	Teacher	kasun@gmail.com	1234	Officer	Teaching
*	HULL	HULL	HULL	HULL	HULL	HULL	HULL	HULL	HULL

Figure 62 - Execute Screenshot of the user_details table result



This screenshot shows the MySQL Workbench interface with the 'MySQL Queries' tab active. The code pane displays the creation of the 'complaints' table:

```

15    PRIMARY KEY (UserId)
16 );
17
18
19 • SELECT * FROM slbfe.complaints;
20
21 • CREATE TABLE `slbfe`.`complaints` (
22     ComplaintId int NOT NULL AUTO_INCREMENT,
23     Complaint varchar(1000) NOT NULL,
24     Reply varchar(1000) NULL,
25    PRIMARY KEY (ComplaintId)
26 );
27
28
29

```

The results grid below shows the data for the 'complaints' table:

ComplaintId	Complaint	Reply
1	Very bad	Thank you
2	Bad	Thanks
*	HULL	HULL

Figure 63 - Execute Screenshot of the complaints table result

MySQL Workbench

Local instance MySQL80 ×

File Edit View Query Database Server Tools Scripting Help

Navigator crudoperationtable MYSQL Queries ×

SCHEMAS

Filter objects

slbfe

Tables

admin

Columns

- AdminId
- UserName
- Password

Indexes

Foreign Keys

Triggers

complaints

user_details

Administration Schemas

Information:

Table: admin

Columns:

AdminId	int AI PK
UserName	varchar(1000)
Password	varchar(1000)

crudoperationtable MYSQL Queries ×

29

30 • `SELECT * FROM slbfe.admin;`

31

32 • `CREATE TABLE `slbfe`.`admin` (`

33 `AdminId int NOT NULL AUTO_INCREMENT,`

34 `UserName varchar(1000) NOT NULL,`

35 `Password varchar(1000) NULL,`

36 `PRIMARY KEY (AdminId)`

37 `);`

38

39

40 • `INSERT INTO `slbfe`.`admin``

41 `(`AdminId`,`

42 ``UserName`,`

43 ``Password`)`

44 `VALUES`

45 `(1,`

46 `'AdminSLBFE',`

47 `'Admin123');`

48

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Co

AdminId	UserName	Password
1	AdminSLBFE	Admin123
NULL	NULL	NULL

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.

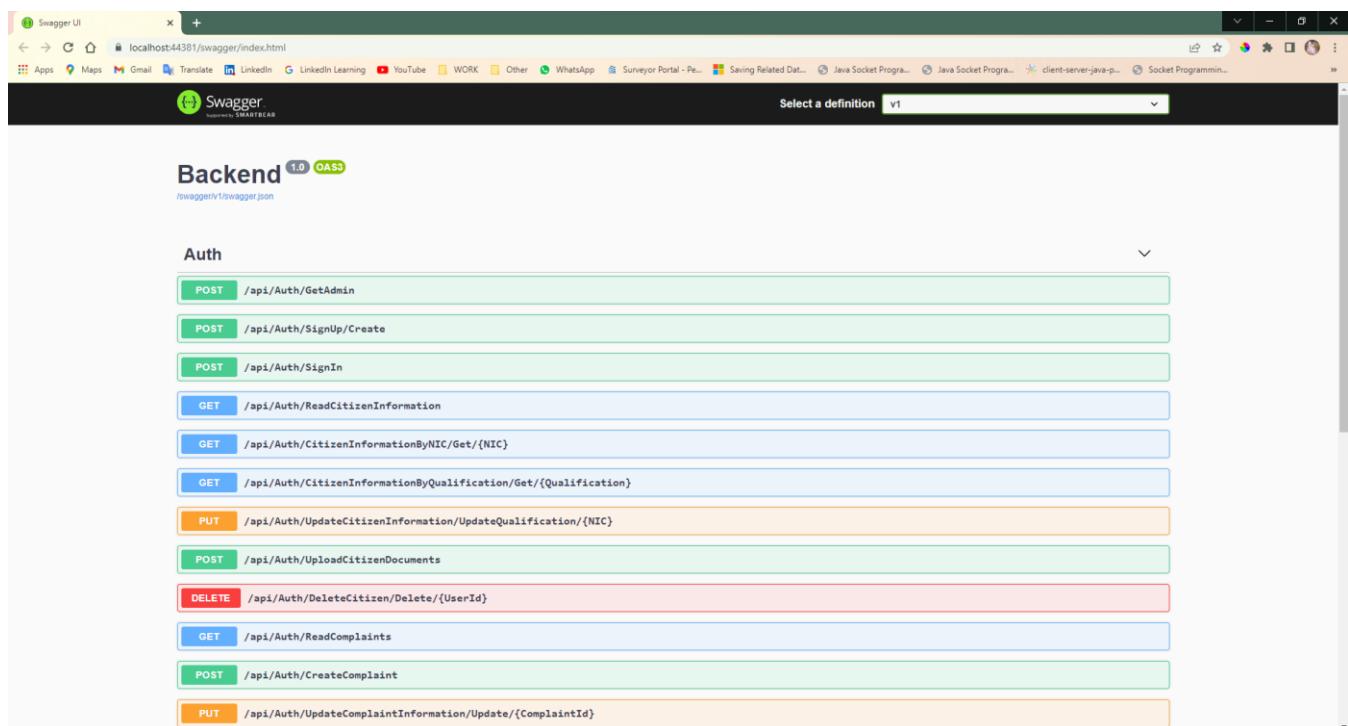
3.2. Platform – Mobile Platform

- 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 officers to reply to the complaints posted by the citizens.



The screenshot shows a browser window displaying the Swagger UI for a Backend API. The title bar says "Swagger UI" and the address bar shows "localhost:44381/swagger/index.html". The main content area is titled "Backend 1.0 OAS". It lists various API endpoints under the "Auth" category, each with its method, URL, and a brief description. The endpoints include:

- POST /api/Auth/GetAdmin
- POST /api/Auth/SignUp/Create
- POST /api/Auth/SignIn
- GET /api/Auth/ReadCitizenInformation
- GET /api/Auth/CitizenInformationByNIC/Get/{NIC}
- GET /api/Auth/CitizenInformationByQualification/Get/{Qualification}
- PUT /api/Auth/UpdateCitizenInformation/UpdateQualification/{NIC}
- POST /api/Auth/UploadCitizenDocuments
- DELETE /api/Auth/DeleteCitizen/Delete/{UserId}
- GET /api/Auth/ReadComplaints
- POST /api/Auth/CreateComplaint
- PUT /api/Auth/UpdateComplaintInformation/Update/{ComplaintId}

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 delete function for the admin. And also, these all fields are validated.

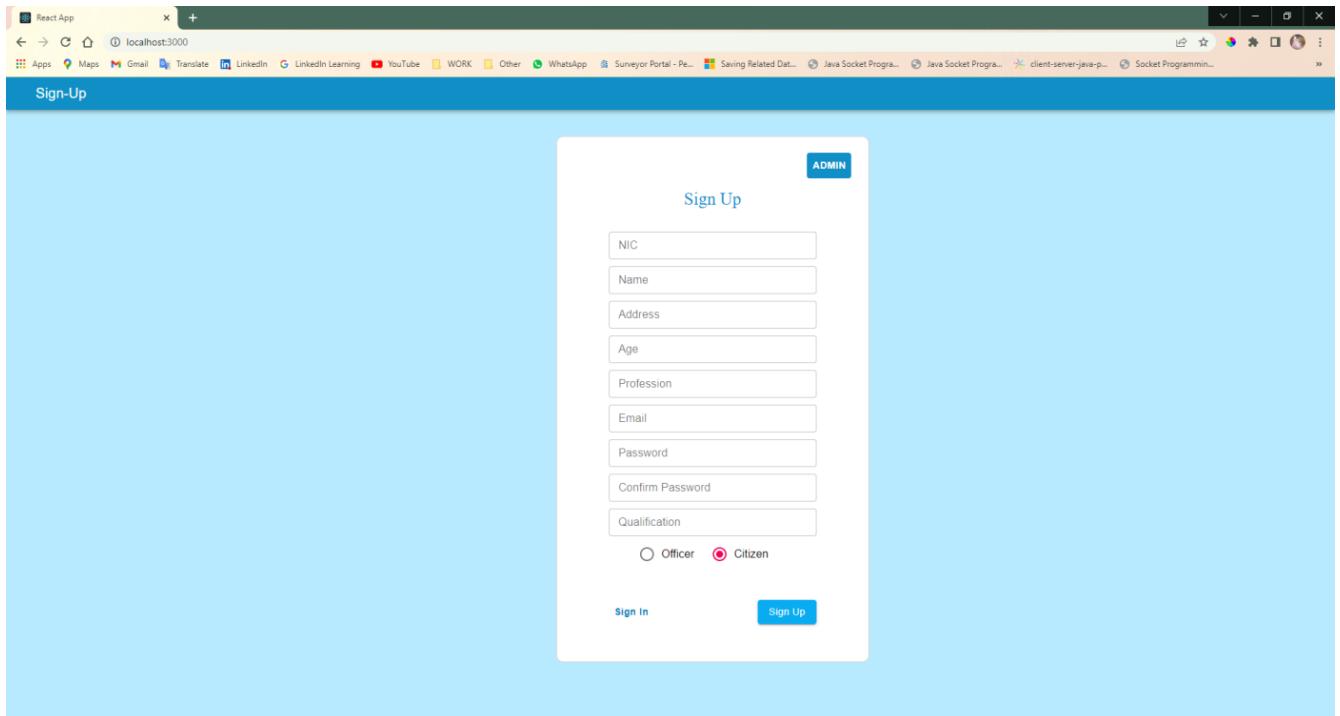


Figure 66- The Frontend Part (Citizen & Office: SignUp)

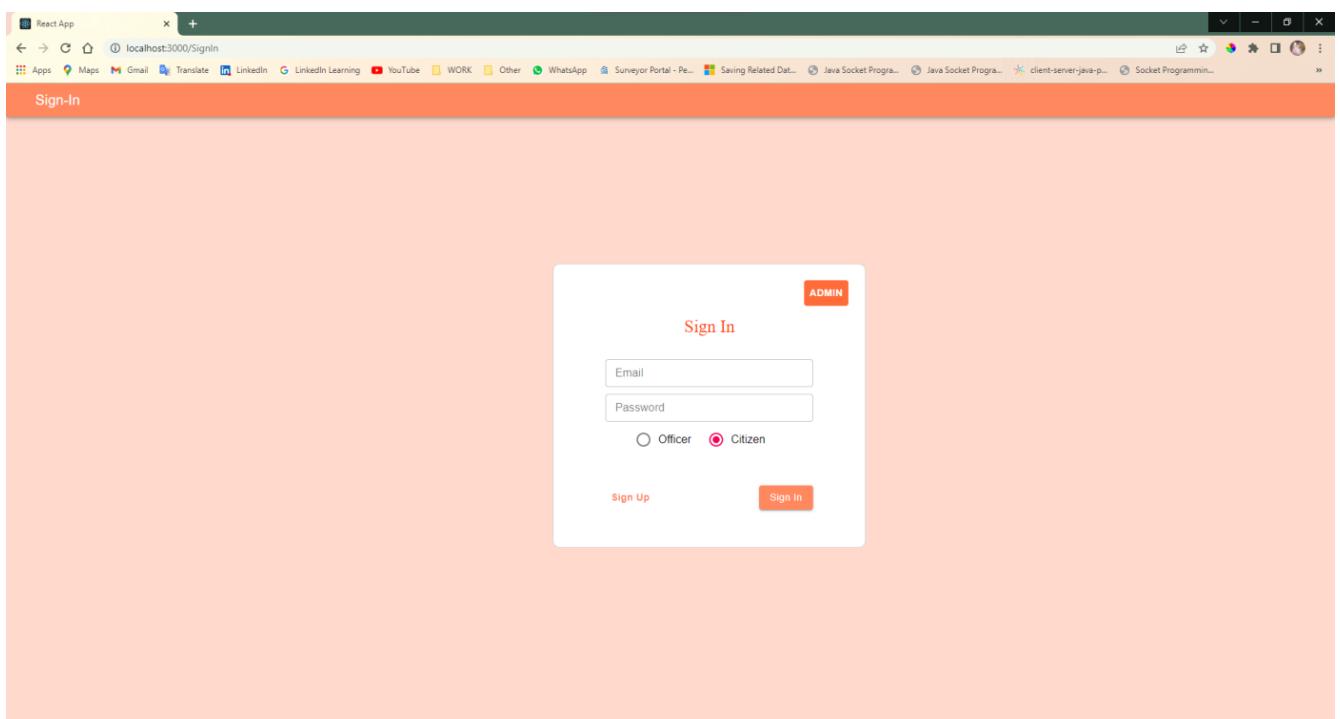


Figure 67 - The Frontend Part (Citizen & Office: SignIn)

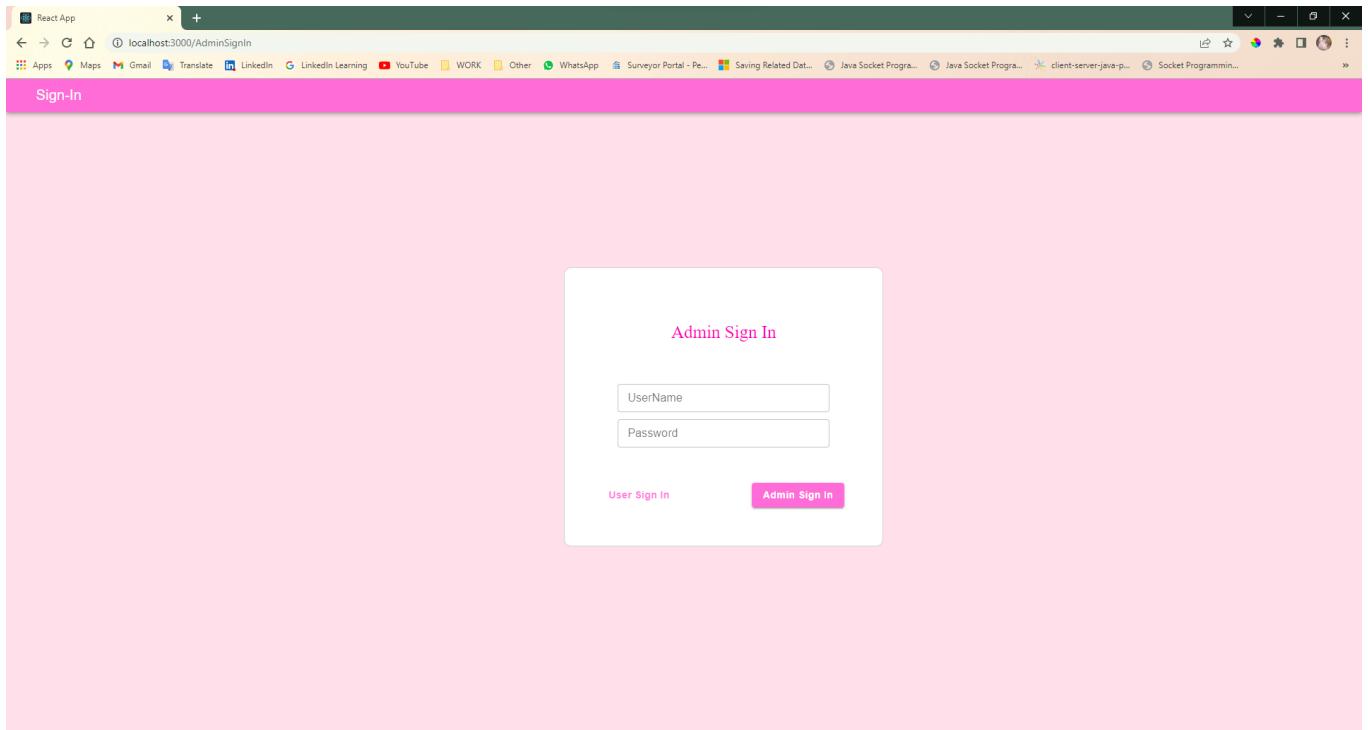


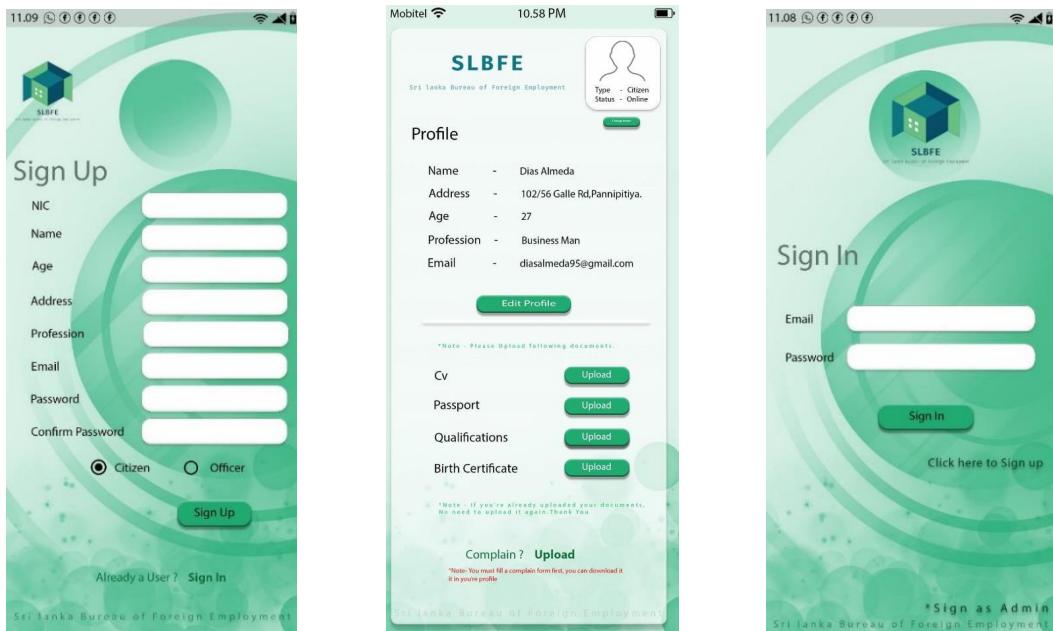
Figure 68 - The Frontend Part (Admin SignIn)

A screenshot of a web browser window titled "React App". The address bar shows "localhost:3000/AdminHomePage". The page has a blue header bar with the text "Admin Home Page". Below it is a green button labeled "List of Citizens". A table follows, with columns: Userid, NIC, Name, Address, Age, Profession, Email, Qualification, and Action. The table contains four rows of citizen information.

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.

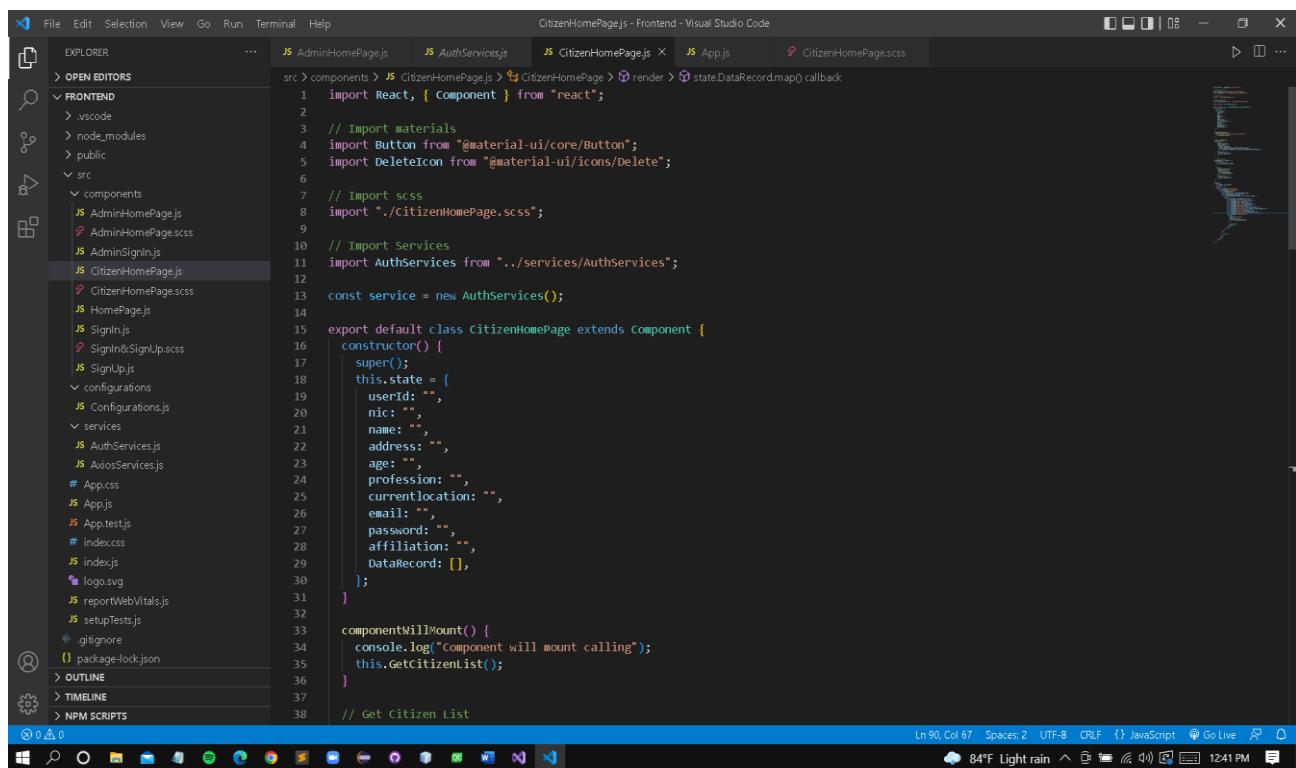
All the officer credentials are stored to the database by POST API and I used to retrofit 2 service for the API communication. All officers' inputs and outputs are handled by APIs. From POST method all officers' inputs store into the database and I used GET API for sign in process. From that method, 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



The screenshot shows the Visual Studio Code interface with the following details:

- File Explorer:** Shows the project structure with files like AdminHomePage.js, AuthService.js, CitizenHomePage.js, App.js, and CitizenHomePage.scss.
- Editor:** The CitizenHomePage.js file is open, displaying React component code. The code imports React, material-ui components, and AuthService. It defines a CitizenHomePage component with state variables for user ID, NIC, name, address, age, profession, email, password, affiliation, and DataRecord. It includes a constructor, componentWillMount, and render methods.
- Bottom Status Bar:** Shows line 90, column 67, spaces: 2, UTF-8, CRLF, JavaScript, Go Live, and system icons for weather (84°F), battery, signal, and time (12:41 PM).

The screenshot shows the Visual Studio Code interface with the CitizenHomePage.js file open in the editor. The code is a class-based component for managing citizen data. It includes methods for getting a citizen list, handling citizen deletion, and rendering the data. The code uses promises and the .map() method to process the data record.

```
// 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>
      </div>
    </div>
  );
}
```

This screenshot shows the same Visual Studio Code session with a specific line of code highlighted in yellow. The line is part of the map function, specifically the return statement where the data record is mapped into individual card components. The rest of the code is identical to the previous screenshot.

```
// 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>
      </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

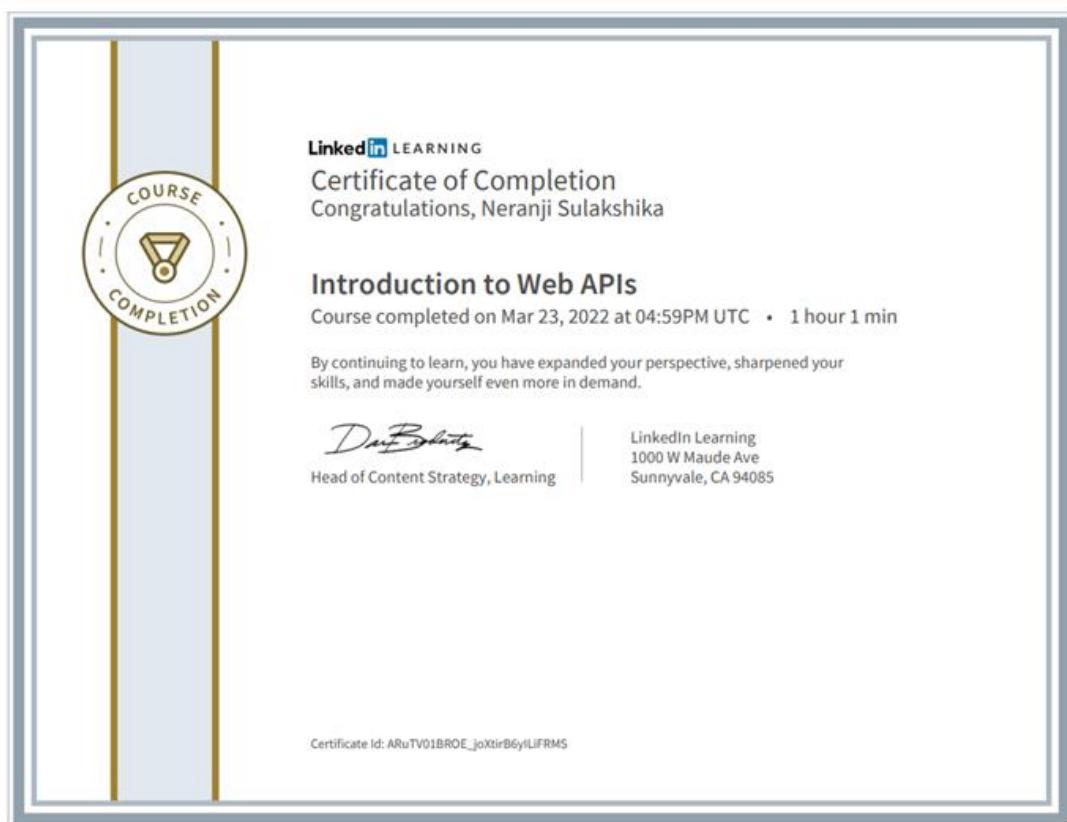


Figure 70 - LinkedIn certificate: 10707385

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

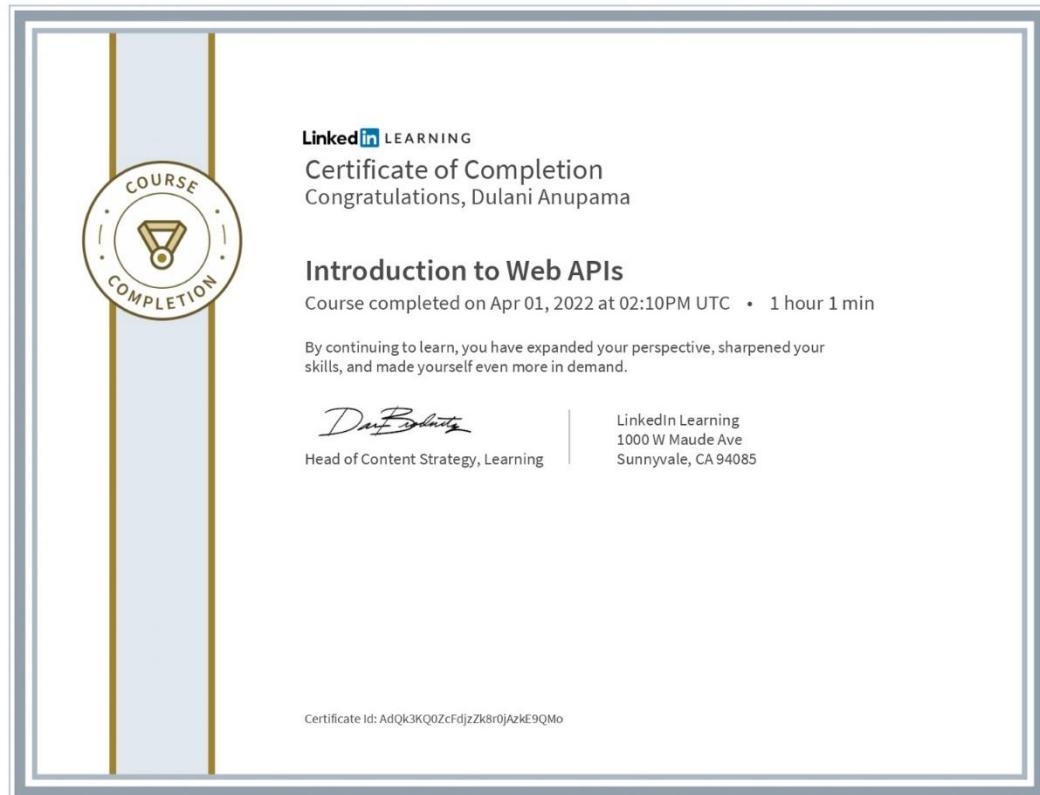


Figure 73 - LinkedIn certificate: 10707129

5.5. 10707372 - P D S Sigera



Figure 74 - LinkedIn certificate: 10707372

5.6. 10707291 – S S N S Nevins

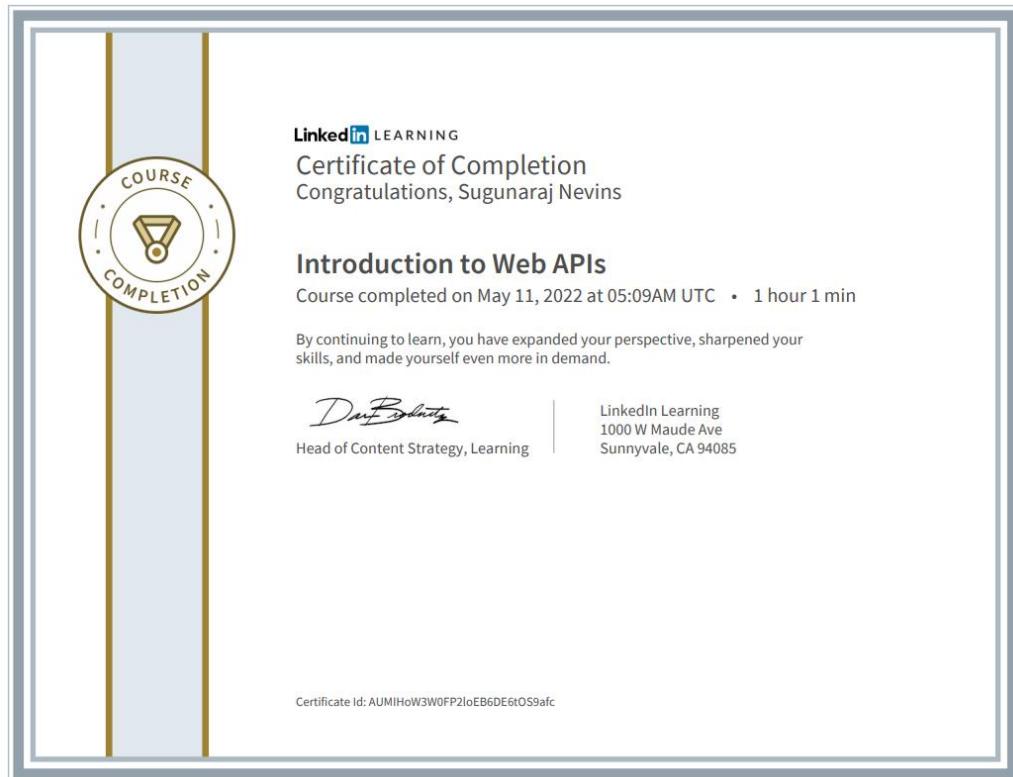


Figure 75 - LinkedIn certificate: 10707291

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.