CS6106

DATABASE MANAGEMENT SYSTEM

PROJECT REPORT

BLOOD DONATION MANAGEMENT SYSTEM

JANANI A
2022503502
TEJESSHREE S
2022503524

Scenario for dB design:

- Blood Bank and Hospital are planning to implement a blood donation management system that supports both users' requirements and gives connectivity.
- Stores Donor's details, Patient's, Blood bank and Hospital details, and Nurse and Blood bank manager details.
- Keeps track of the availability of blood units.
- Keeps track of the donor-to-patient transaction path
- Communication logs

Tools Study

Tool used: Microsoft SQL Server

Backend:

1. **ASP.NET Core Web API:**

Framework: ASP.NET Core

Purpose: Provides the server-side logic and functionality for your application.

Controllers and Classes: Implemented to handle the logic for each table in your database (e.g., handling CRUD operations).

Operations: Includes GET, POST, PUT, and DELETE operations to manage data.

Database Connection: Uses Entity Framework Core to interact with Microsoft SQL Server.

2. **Microsoft SQL Server:**

Database: blooddon1

Tables: Contains structured data relevant to your application (e.g., donor information, blood types, donation records).

Function: Stores and retrieves data as requested by the ASP.NET Core Web API.

Frontend

3. **React.js:**

Library: React.js, a popular JavaScript library for building user interfaces.

Purpose: Provides the client-side logic and the user interface.

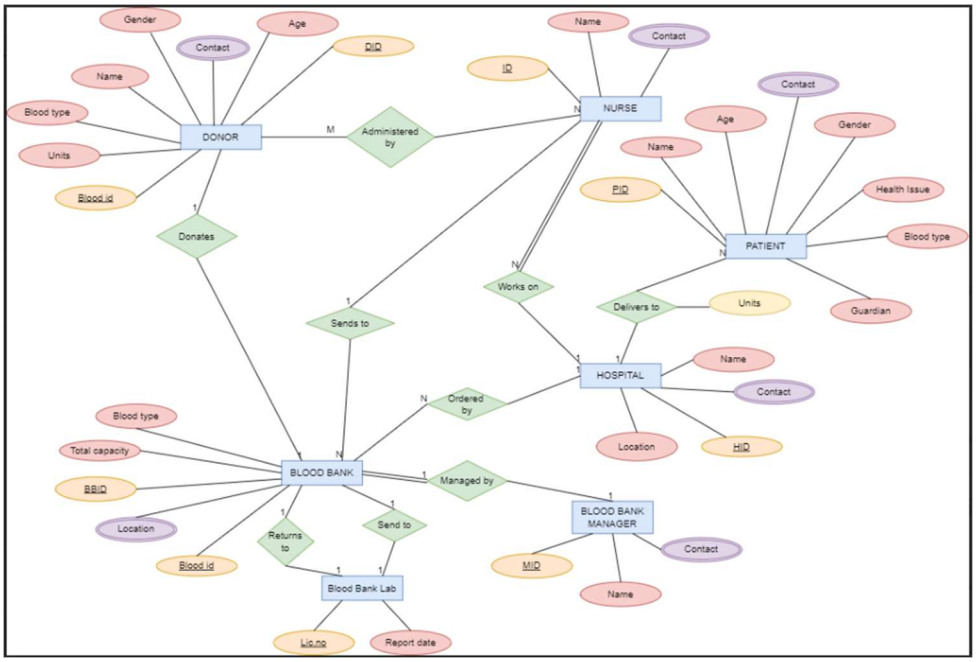
Components: Built using React components to create a dynamic and responsive web application.

Communication with Backend: Utilizes asynchronous HTTP requests (such as fetch or Axios) to interact with the ASP.NET Core Web API, performing CRUD operations.

Reason for choosing SQL Server

- SQL Server is a Relational Database Management software that helps structure our database.
- Ensures integrity and reliability with constraints
- Supports scalability, user-friendly interface
- Comprehensive security features - encryption, access controls, etc.
- Allows for integration with other applications to enable workflow automation (SSIS)
- Integration with communication systems like SMS, and email enables automated notifications
- Integration with EHR (Electronic Health Record)
- Backup and restoring capabilities
- SSRS (SQL Server Reporting Services), SSAS (Analysis services).

ER Diagram



Attributes

- Donor
- Nurse
- Blood Bank
- Blood Bank Lab
- Blood Bank Manager
- Patient
- Hospital

Relationship

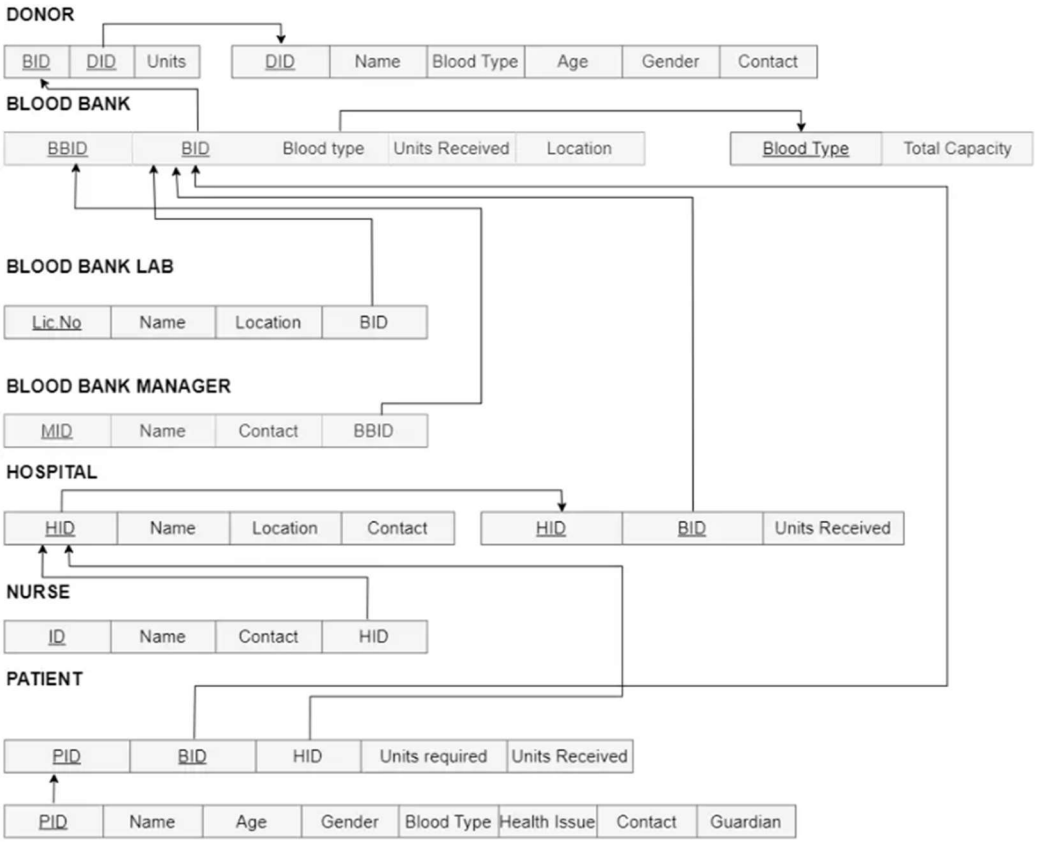
- Donor administered by Nurse
- Donor donates to Blood Bank
- Nurse sends Blood to Blood Bank
- Blood Bank sends the donated blood to Blood Bank Lab
- Blood Bank managed by Blood Bank Manager
- Hospital orders Blood from Blood Bank
- Hospital delivers Blood to the Patient
- Nurse works for the Hospital

Primary Keys

- BID- Blood ID
- BBID- Blood Bank ID
- HID- Hospital ID
- DID- Donor ID
- MID- Manager ID
- PID- Patient ID

Relational Schema

The Relational Schema is refined to **3NF**.

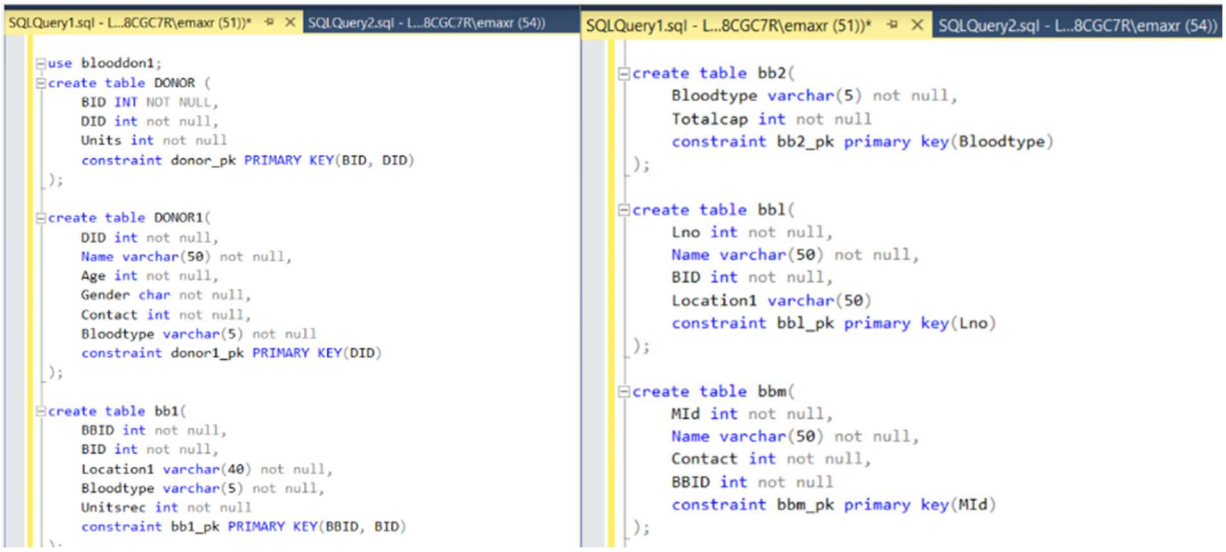


TABLES CREATION

The tables are created using SQL queries with NOT NULL and PRIMARY KEY constraints.

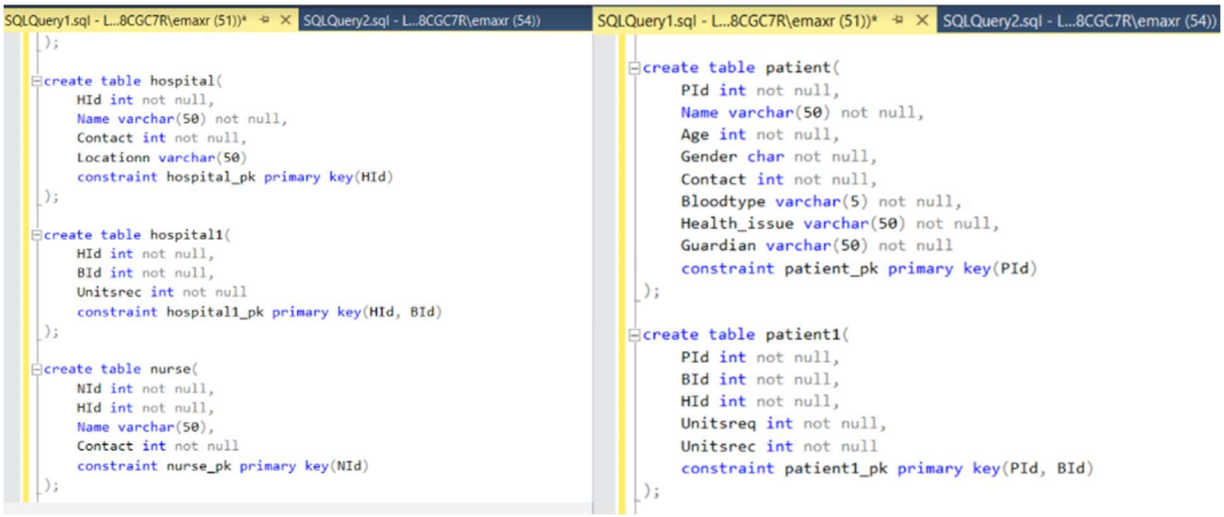
At first, the database named "**BLOODDON1"** is created. Under the database, tables for the donor, blood bank, blood bank lab, blood bank manager, Nurse, Hospital, and patients are created.

DDL Commands



```
use blooddon1;
create table DONOR (
    BID INT NOT NULL,
    DID int not null,
    Units int not null
    constraint donor_pk PRIMARY KEY(BID, DID)
);

create table DONOR1(
    DID int not null,
    Name varchar(50) not null,
    Age int not null,
    Gender char not null,
    Contact int not null,
    Bloodtype varchar(5) not null
    constraint donor1_pk PRIMARY KEY(DID)
);

create table bb1(
    BBID int not null,
    BID int not null,
    Location1 varchar(40) not null,
    Bloodtype varchar(5) not null,
    Unitsrec int not null
    constraint bb1_pk PRIMARY KEY(BBID, BID)
```

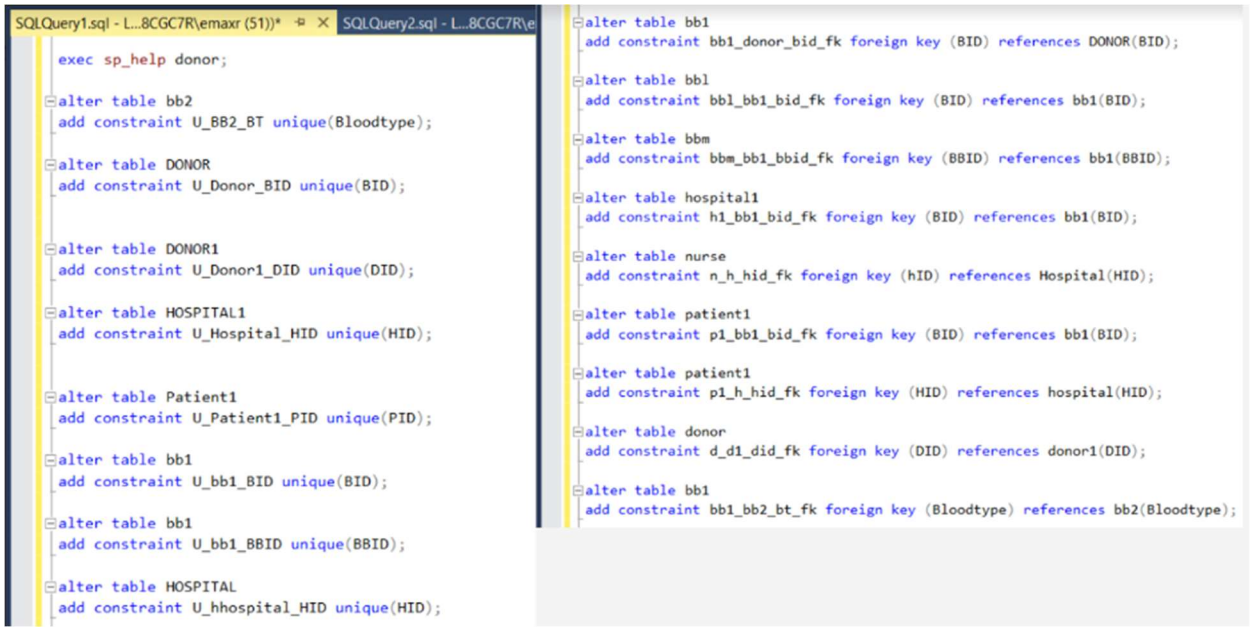
```
create table bb2(
    Bloodtype varchar(5) not null,
    Totalcap int not null
    constraint bb2_pk primary key(Bloodtype)
);

create table bbl(
    Lno int not null,
    Name varchar(50) not null,
    BID int not null,
    Location1 varchar(50)
    constraint bbl_pk primary key(Lno)
);

create table bbm(
    MId int not null,
    Name varchar(50) not null,
    Contact int not null,
    BBID int not null
    constraint bbm_pk primary key(MId)
);
```



```
);

create table hospital(
    HId int not null,
    Name varchar(50) not null,
    Contact int not null,
    Locationn varchar(50)
    constraint hospital_pk primary key(HId)
);

create table hospital1(
    HId int not null,
    BId int not null,
    Unitsrec int not null
    constraint hospital1_pk primary key(HId, BId)
);

create table nurse(
    NId int not null,
    HId int not null,
    Name varchar(50),
    Contact int not null
    constraint nurse_pk primary key(NId)
);
```

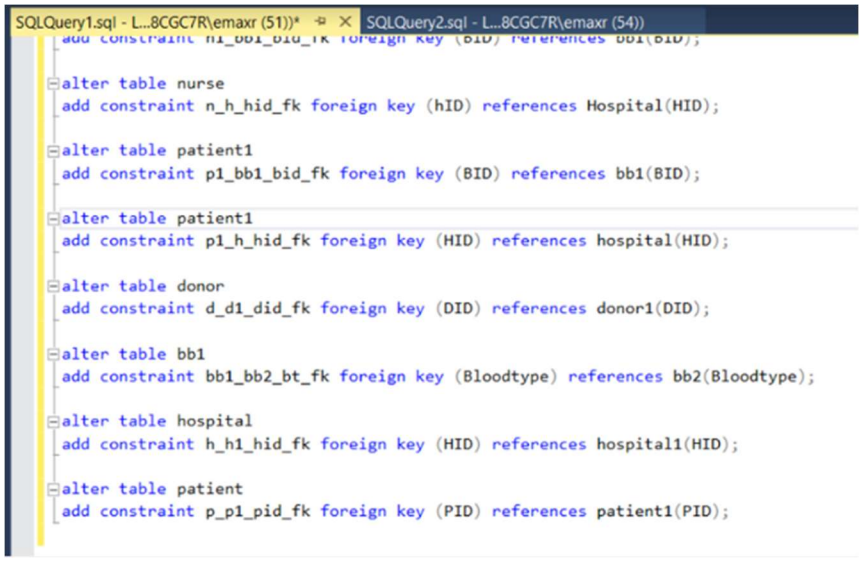
```
create table patient(
    PId int not null,
    Name varchar(50) not null,
    Age int not null,
    Gender char not null,
    Contact int not null,
    Bloodtype varchar(5) not null,
    Health_issue varchar(50) not null,
    Guardian varchar(50) not null
    constraint patient_pk primary key(PId)
);

create table patient1(
    PId int not null,
    BId int not null,
    HId int not null,
    Unitsreq int not null,
    Unitsrec int not null
    constraint patient1_pk primary key(PId, BId)
);
```

The other constraints like the FOREIGN key and UNIQUE key constraints are added using the ALTER command.



```
SQLQuery1.sql - L...8CGC7R\emaxr (51))*   SQLQuery2.sql - L...8CGC7R\e

exec sp_help donor;

alter table bb2
add constraint U_BB2_BT unique(Bloodtype);

alter table DONOR
add constraint U_Donor_BID unique(BID);


alter table DONOR1
add constraint U_Donor1_DID unique(DID);

alter table HOSPITAL1
add constraint U_Hospital_HID unique(HID);


alter table Patient1
add constraint U_Patient1_PID unique(PID);

alter table bb1
add constraint U_bb1_BID unique(BID);

alter table bb1
add constraint U_bb1_BBID unique(BBID);

alter table HOSPITAL
add constraint U_hhospital_HID unique(HID);
```

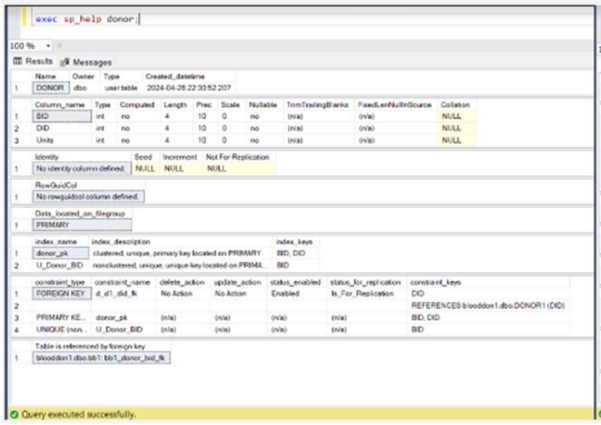
```
alter table bb1
add constraint bb1_donor_bid_fk foreign key (BID) references DONOR(BID);

alter table bbl
add constraint bbl_bb1_bid_fk foreign key (BID) references bb1(BID);

alter table bbm
add constraint bbm_bb1_bbid_fk foreign key (BBID) references bb1(BBID);

alter table hospital1
add constraint h1_bb1_bid_fk foreign key (BID) references bb1(BID);

alter table nurse
add constraint n_h_hid_fk foreign key (hID) references Hospital(HID);

alter table patient1
add constraint p1_bb1_bid_fk foreign key (BID) references bb1(BID);

alter table patient1
add constraint p1_h_hid_fk foreign key (HID) references hospital(HID);

alter table donor
add constraint d_d1_did_fk foreign key (DID) references donor1(DID);

alter table bb1
add constraint bb1_bb2_bt_fk foreign key (Bloodtype) references bb2(Bloodtype);
```



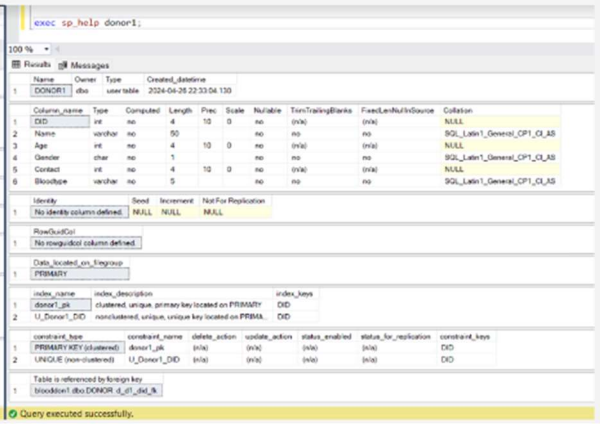
```
SQLQuery1.sql - L...8CGC7R\emaxr (51))*   SQLQuery2.sql - L...8CGC7R\emaxr (54))

alter table nurse
add constraint n_h_hid_fk foreign key (hID) references Hospital(HID);

alter table patient1
add constraint p1_bb1_bid_fk foreign key (BID) references bb1(BID);

alter table patient1
add constraint p1_h_hid_fk foreign key (HID) references hospital(HID);

alter table donor
add constraint d_d1_did_fk foreign key (DID) references donor1(DID);

alter table bb1
add constraint bb1_bb2_bt_fk foreign key (Bloodtype) references bb2(Bloodtype);

alter table hospital
add constraint h_h1_hid_fk foreign key (HID) references hospital1(HID);

alter table patient
add constraint p_p1_pid_fk foreign key (PID) references patient1(PID);
```

OUTPUT

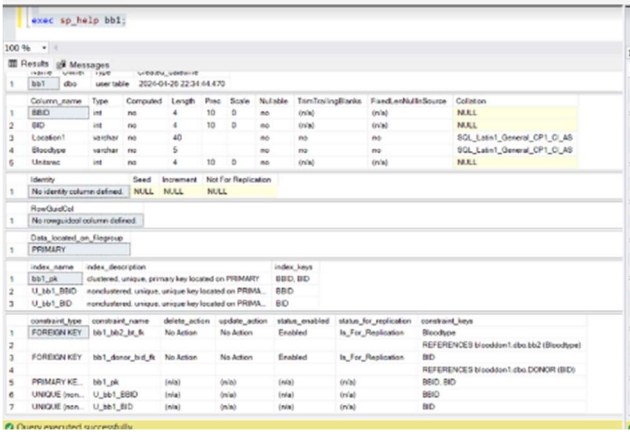
DONOR



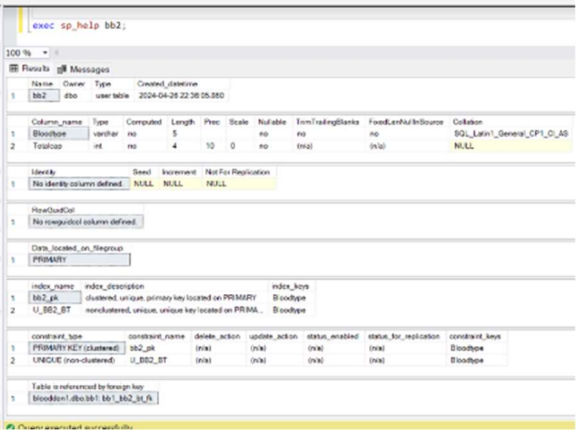
DONOR1



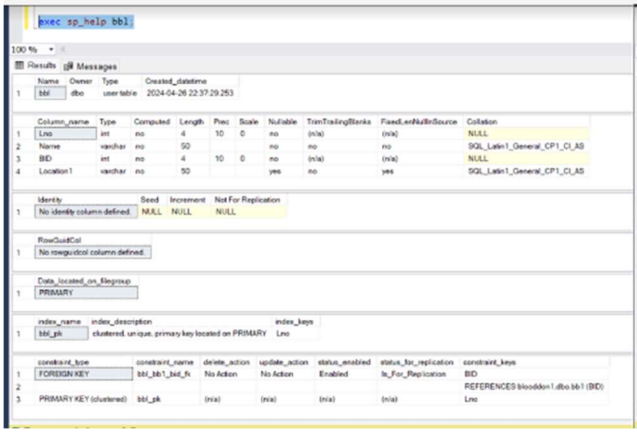
BB1 (BloodBank 1)



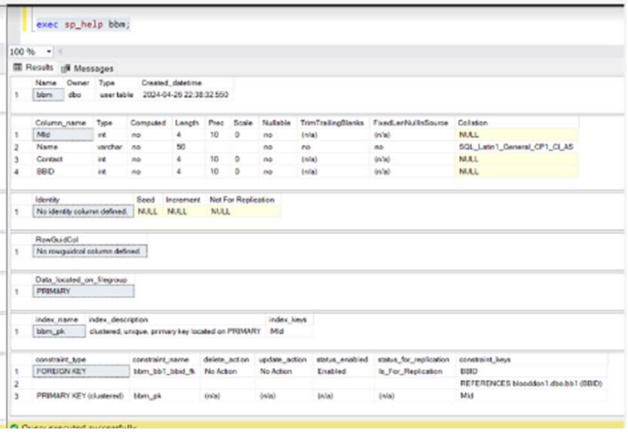
BB2 (Blood Bank 2)



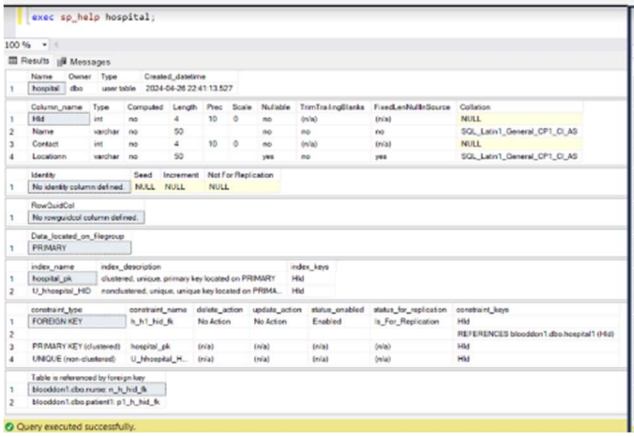
BBL (Blood Bank Lab)



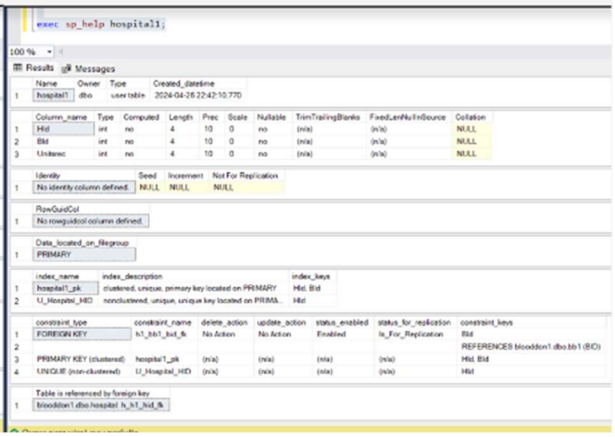
BBM (Blood Bank manager)



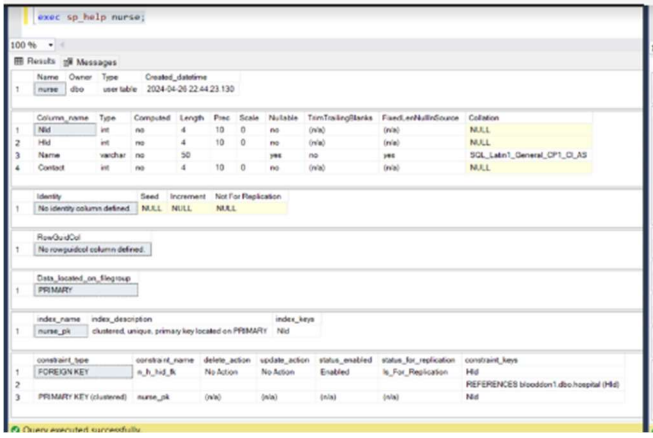
HOSPITAL



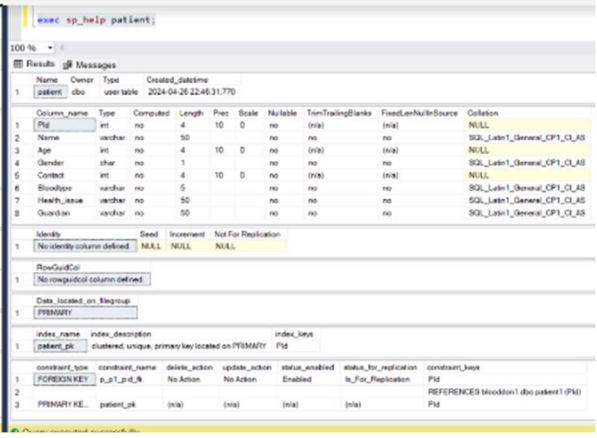
HOSPITAL1



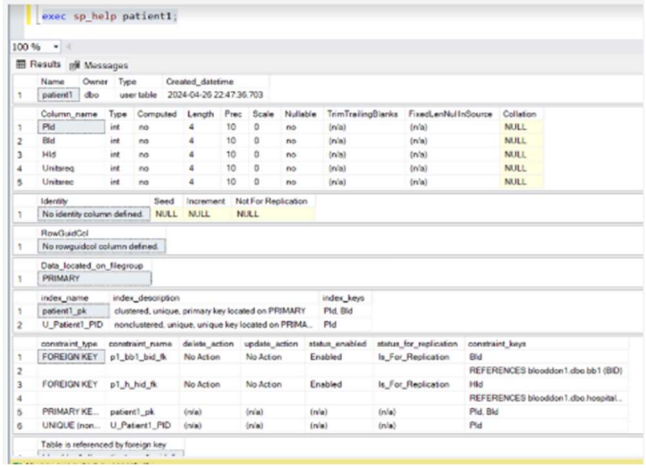
NURSE



PATIENT



PATIENT1



DML Commands

The insertion on tables is done using the multiple row insert command. The primary key column entries are inserted followed by other column entries.

The insertion queries implemented in the SQL server and their Output are listed below.

DONOR1

```
insert into donor1 (DID, Name, Age, Gender, Contact, Bloodtype) values
(101, 'Mark', 40, 'M', 987654321, 'O+'),
(102, 'Rose', 38, 'F', 9876555551, 'A+'),
(103, 'Jenny', 55, 'F', 987654221, 'B+'),
(104, 'Park', 24, 'M', 987654311, 'B+'),
(105, 'Mary', 34, 'F', 777654321, 'A-');
```

100 %

Results | Messages

	DID	Name	Age	Gender	Contact	Bloodtype
1	101	Mark	40	M	987654321	O+
2	102	Rose	38	F	9876555551	A+
3	103	Jenny	55	F	987654221	B+
4	104	Park	24	M	987654311	B+
5	105	Mary	34	F	777654321	A-

DONOR

```
insert into donor(BID, DID, Units) values
(901, 101, 5),
(902, 102, 4),
(903, 102, 3),
(904, 105, 2);

select * from donor;
```

100 %

Results | Messages

	BID	DID	Units
1	901	101	5
2	902	102	4
3	903	102	3
4	904	105	2

BB2(Blood bank 2)

```
insert into bb2 (Bloodtype, Totalcap) values
('O+', 0),
('A+', 4),
('B+', 0),
('O-', 0),
('A-', 0),
('B-', 0),
('AB+', 0),
('AB-', 0);

select * from bb2;
```

100 %

Results | Messages

	Bloodtype	Totalcap
1	A+	4
2	A-	0
3	AB+	0
4	AB-	0
5	B+	0
6	B-	0
7	O+	0
8	O-	0

BB1(Blood bank 1)

```
insert into bb1 (BBID, BID, Location1, Bloodtype, Unitsrec) values
(701, 901, 'Chennai', 'O+', 2),
(702, 904, 'Bangalore', 'A-', 1),
(703, 902, 'Chennai', 'A+', 2),
(704, 903, 'Pune', 'A+', 2);

select * from bb1;
```

100 %

Results | Messages

	BBID	BID	Location1	Bloodtype	Unitsrec
1	701	901	Chennai	O+	2
2	702	904	Bangalore	A-	1
3	703	902	Chennai	A+	2
4	704	903	Pune	A+	2

BLOOD BANK LAB (BBL)

```
insert into bbl (Lno, BID, Location1) values
(301, 901, 'Chennai'),
(302, 901, 'Salem'),
(303, 902, 'Coimbatore'),
(304, 903, 'Hosur');

select * from bbl;
```

100 %

Results | Messages

	Lno	BID	Location1
1	301	901	Chennai
2	302	901	Salem
3	303	902	Coimbatore
4	304	903	Hosur

RETRIEVAL OF DATUM

A few retrieval queries like **select, aggregate, group by** are performed to check the database is created and implemented without any error**.**

Query

List the details of donor whose blood type is B+.

Command and Output

```
select * from donor1 where bloodtype='B+';
```

.00 %

Results | Messages

	DID	Name	Age	Gender	Contact	Bloodtype
1	103	Jenny	55	F	987654221	B+
2	104	Park	24	M	987654311	B+

Query

List the total units received by the blood bank for each blood type available.

Command and output

```
--List total units recieved by bloodbank for each bloodtype available
select sum(unitsrec) as Total_Units_Recieved, bloodtype from bb1 group by (bloodtype);
```

100 %

Results | Messages

	Total_Units_Recieved	bloodtype
1	5	A+
2	1	A-
3	2	O+

Query

List all the donor ID, blood ID, and units donated by donors whose blood type is A+.

Command and output

```
--List DID, BID, Units donated by donor whose bloodtype is A+
select d1.DID, d.BID, d.Units from donor d join donor1 d1 on d.DID=d1.DID where d1.Bloodtype='A+';
```

100 %

Results | Messages

	DID	BID	Units
1	102	902	4
2	102	903	3

PL/SQL- Trigger function is used on the table **Blood bank 2 (BB2).** The insertion, deletion, or updation of the Blood Bank 1 table is triggered and the BB2 table is updated.

```
--update trigger
CREATE TRIGGER trgAfterUpdate ON bb1
AFTER UPDATE
AS
BEGIN
    SET NOCOUNT ON;
    -- Debug message
    PRINT 'update trigger fired';
    UPDATE bb2
    SET Totalcap = Totalcap - d.Unitsrec + i.Unitsrec
    FROM bb2 b
    JOIN deleted d ON b.Bloodtype = d.Bloodtype
    JOIN inserted i ON b.Bloodtype = i.Bloodtype;
    -- Debug message
    PRINT 'Update to bb2 completed';
END;

UPDATE bb1 SET Unitsrec = 3 WHERE BBID = 703;
SELECT * FROM bb2;
```

100 %

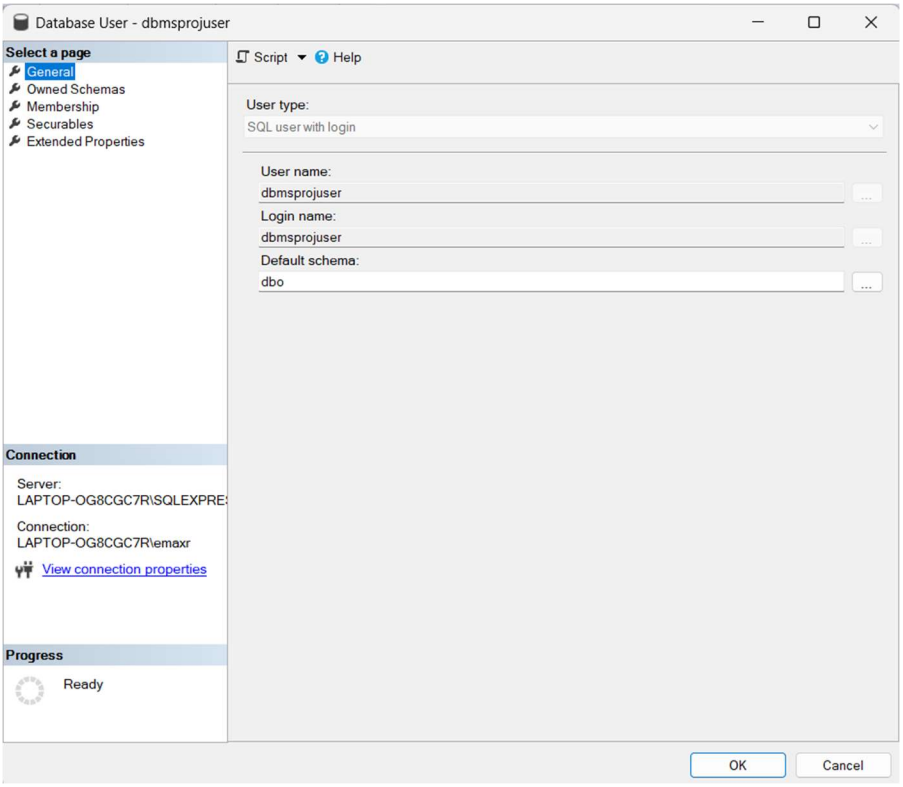
Results | Messages

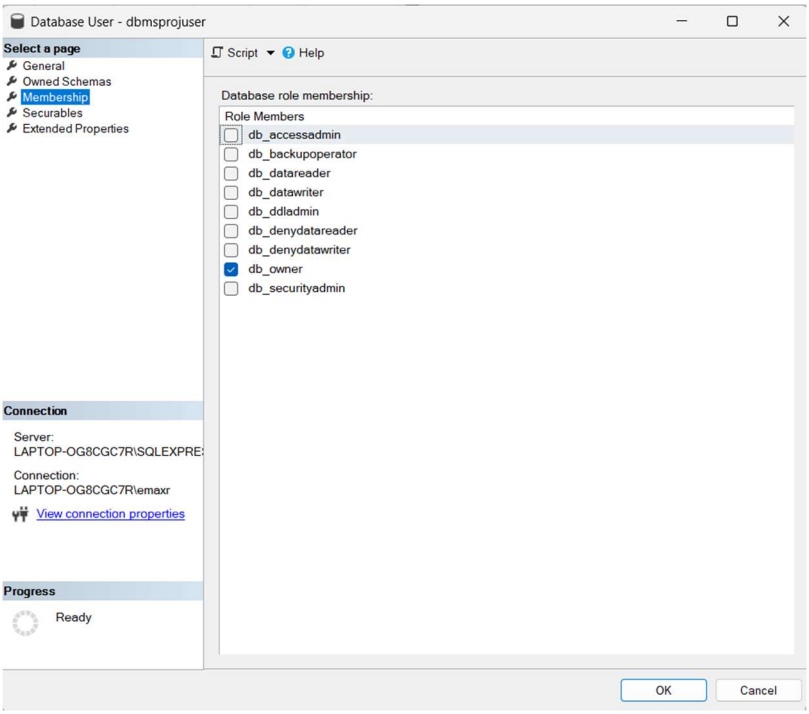
	Bloodtype	Totalcap
1	A+	5
2	A-	0
3	AB+	0
4	AB-	0
5	B+	0
6	B-	0
7	O+	0
8	O-	0

Query executed successfully.

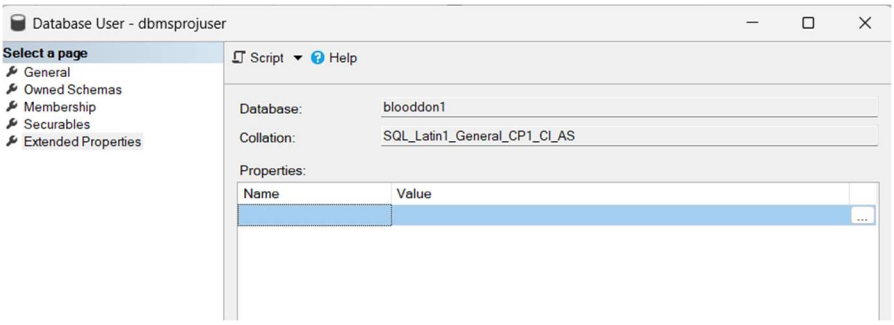
Creating users with roles on the database blooddon1 in MS SQL server:

The login name "**dbmsprojuser"** and password are created, and settings in Owned Schemas, and Membership are made under the SQL server EXPRESS that is used to connect the database with the backend.

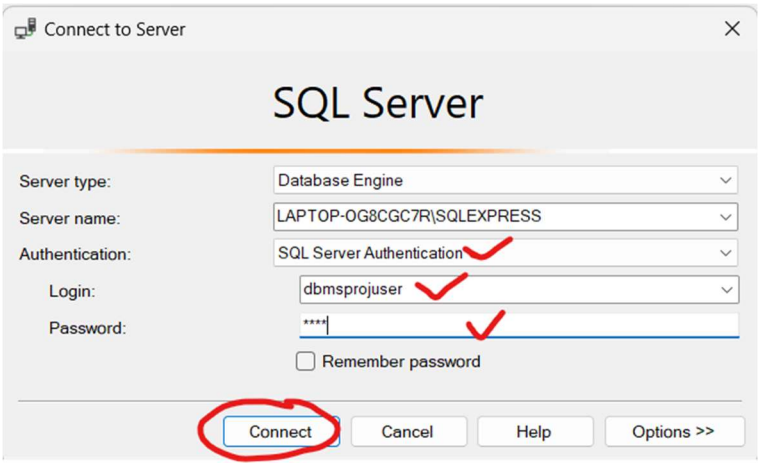




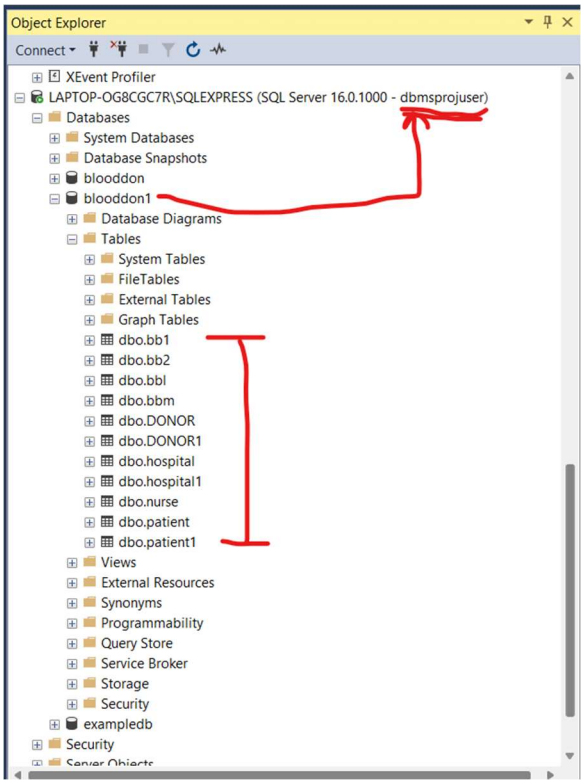
The user is assigned the roles on the database ‘Blooddon1’.



The local host is connected to the server and the authentication is made as “SQL Server Authentication”. The login name and password are given so that the database connects to the backend.

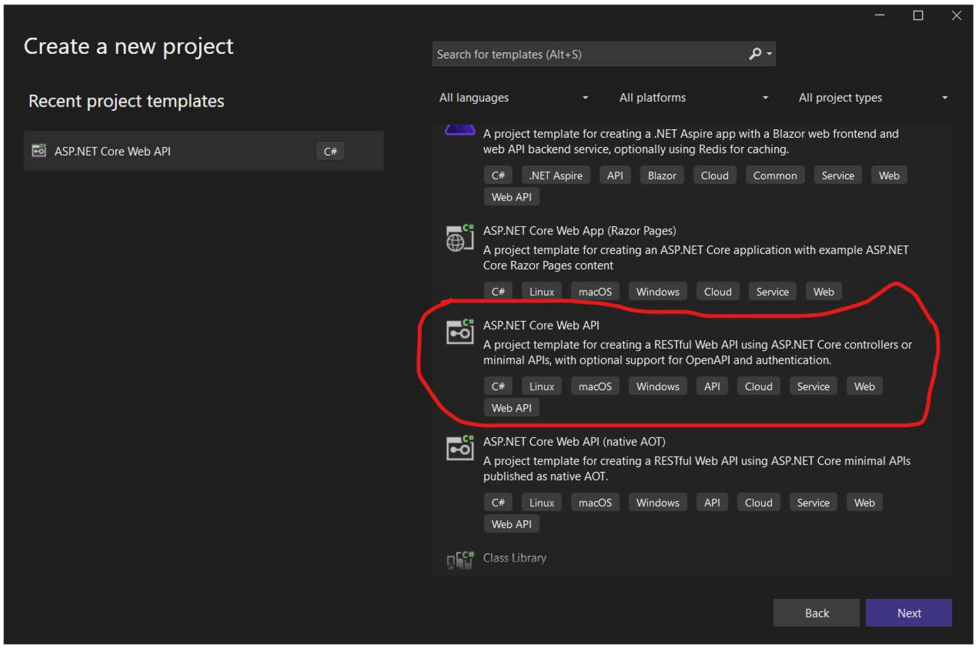


The database "**blooddon1**" is assigned to the "**dbmsprojuser**" and all the tables are accessed such that it can be connected to the backend and frontend processes.



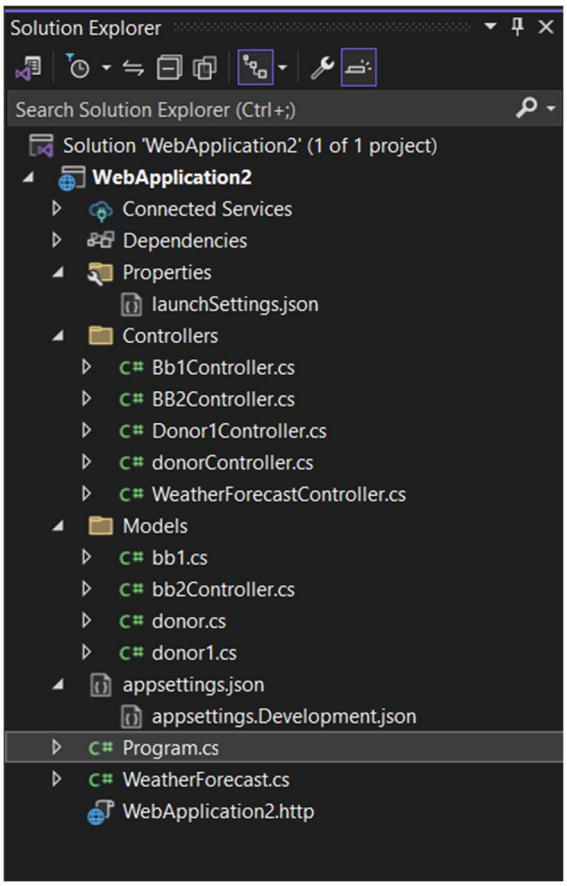
Creating ASP.NET Core Web API in Visual Studio:

A new project of the template ASP.NET Core Web API is created.



The following is the list of C# files in the project, where the Program.cs file creates the web host which helps the app to listen to HTTP requests.

Appsettings.Development.json file contains the configuration details like the database details, and the connection strings.



Program.cs

```
using Microsoft.AspNetCore.Mvc.Formatters;
using Microsoft.Extensions.Options;
using Newtonsoft.Json.Serialization;


var builder = WebApplication.CreateBuilder(args);

// Add services to the container.

builder.Services.AddControllers(options =>
{
    // Configure Newtonsoft.Json options here
    options.OutputFormatters.RemoveType<SystemTextJsonOutputFormatter>();
}).AddNewtonsoftJson(options =>
{
    // Configure Newtonsoft.Json serializer settings here
    options.SerializerSettings.ReferenceLoopHandling =
Newtonsoft.Json.ReferenceLoopHandling.Ignore;
    options.SerializerSettings.ContractResolver = new DefaultContractResolver();
});
// Learn more about configuring Swagger/OpenAPI at https://aka.ms/aspnetcore/swashbuckle
builder.Services.AddEndpointsApiExplorer();
```

```
builder.Services.AddSwaggerGen();


// Enable CORS
builder.Services.AddCors(options =>
{
    options.AddPolicy("AllowOrigin",
        builder => builder.AllowAnyOrigin().AllowAnyMethod().AllowAnyHeader());
});

var app = builder.Build();

// Configure the HTTP request pipeline.
if (app.Environment.IsDevelopment())
{
    app.UseSwagger();
    app.UseSwaggerUI();
}

// Enable CORS
app.UseCors("AllowOrigin");

app.UseAuthorization();

app.MapControllers();

app.Run();
```

appsettings.Development.json

```
{
  "ConnectionStrings": {
    "DefaultConnection": "Server=LAPTOP-OG8CGC7R\\SQLEXPRESS;Database=blooddon1; User
ID=dbmsprojuser; Password=1234; Trusted_Connection=False; "
  },

  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft.AspNetCore": "Warning"
    }
  }
}
```

Models: Class files:

Contains the models of the project defined as classes.

Bb1.cs

```
using Microsoft.AspNetCore.Http;
using Microsoft.AspNetCore.Mvc;
namespace WebApplication2.Models
{
public class Bb1
{
public required int BBID { get; set; }
public required int BID { get; set; }
public string Location1 { get; set; }
public string Bloodtype { get; set; }

public int Unitsrec { get; set; }
}
}
```

Bb2:

```
using Microsoft.AspNetCore.Http;
using Microsoft.AspNetCore.Mvc;
namespace WebApplication2.Models
{
public class Bb2
{
public required string BloodType { get; set; }
public int TotalCap { get; set; }
}
}
```

Donor:

```
namespace WebApplication2.Models
{
public class Donor
```

```
{
public int BID { get; set; }
public int DID { get; set; }
public int Units { get; set; }
}
}
```

Donor1:

```
using Microsoft.AspNetCore.Http;
using Microsoft.AspNetCore.Mvc;
namespace WebApplication2.Models
{
public class Donor1
{
public required int DID { get; set; }
public string Name { get; set; }
public int Age { get; set; }
public string Gender { get; set; }
public long Contact { get; set; }
public string Bloodtype { get; set; }
}
}
```

Controllers:

The empty controller API is added for each table details. The contents of each table are copied to the *controller API*. *Get, Post* method is used in the API for selection and insertion. *Put, delete http* requests are included such that updation and deletion are done respectively.

Bb1Controller.cs:

```
using Microsoft.AspNetCore.Http;
using Microsoft.AspNetCore.Mvc;
using Microsoft.Extensions.Configuration;
using System;
using System.Data;
using System.Data.SqlClient;
using WebApplication2.Models;

namespace WebApplication2.Controllers
```

```
{
    [Route("api/[controller]")]
    [ApiController]
    public class Bb1Controller : ControllerBase
    {
        private readonly IConfiguration _configuration;

        public Bb1Controller(IConfiguration configuration)
        {
            _configuration = configuration;
        }

        [HttpGet]
        public IActionResult Get()
        {
            string sqlDataSource = _configuration.GetConnectionString("DefaultConnection");
            if (string.IsNullOrEmpty(sqlDataSource))
            {
                return StatusCode(StatusCodes.Status500InternalServerError, "DefaultConnection is
not defined in the configuration.");
            }

            try
            {
                using (SqlConnection conn = new SqlConnection(sqlDataSource))
                {
                    conn.Open();
                    using (SqlCommand cmd = new SqlCommand("SELECT * FROM bb1", conn))
                    {
                        SqlDataReader reader = cmd.ExecuteReader();
                        DataTable dataTable = new DataTable();
                        dataTable.Load(reader);

                        return Ok(dataTable);
                    }
                }
            }
            catch (SqlException ex)
            {
                return StatusCode(StatusCodes.Status500InternalServerError, ex.Message);
            }
        }

        [HttpPost]
        public IActionResult Post(Bb1 bb1)
        {
            string sqlDataSource = _configuration.GetConnectionString("DefaultConnection");
            if (string.IsNullOrEmpty(sqlDataSource))
            {
                return StatusCode(StatusCodes.Status500InternalServerError, "DefaultConnection is
not defined in the configuration.");
            }

            try
            {
                using (SqlConnection conn = new SqlConnection(sqlDataSource))
                {
                    conn.Open();
                    string query = "INSERT INTO bb1 (BBID, BID, Location1, Bloodtype, Unitsrec)
VALUES (@BBID, @BID, @Location1, @Bloodtype, @Unitsrec)";
                    using (SqlCommand cmd = new SqlCommand(query, conn))
                    {
```

```
                        cmd.Parameters.AddWithValue("@BBID", bb1.BBID);
                        cmd.Parameters.AddWithValue("@BID", bb1.BID);
                        cmd.Parameters.AddWithValue("@Location1", bb1.Location1);
                        cmd.Parameters.AddWithValue("@Bloodtype", bb1.Bloodtype);
                        cmd.Parameters.AddWithValue("@Unitsrec", bb1.Unitsrec);
                        cmd.ExecuteNonQuery();
                    }
                }
                return Ok("Record inserted successfully.");
            }
            catch (SqlException ex)
            {
                return StatusCode(StatusCodes.Status500InternalServerError, ex.Message);
            }
        }

        [HttpPut("{id}")]
        public IActionResult Put(int id, Bb1 bb1)
        {
            string sqlDataSource = _configuration.GetConnectionString("DefaultConnection");
            if (string.IsNullOrEmpty(sqlDataSource))
            {
                return StatusCode(StatusCodes.Status500InternalServerError, "DefaultConnection is
not defined in the configuration.");
            }

            try
            {
                using (SqlConnection conn = new SqlConnection(sqlDataSource))
                {
                    conn.Open();
                    string query = "UPDATE bb1 SET  Location1=@Location1, Unitsrec=@Unitsrec
WHERE id = @Id";
                    using (SqlCommand cmd = new SqlCommand(query, conn))
                    {


                        cmd.Parameters.AddWithValue("@Location1", bb1.Location1);
                        cmd.Parameters.AddWithValue("@Unitsrec", bb1.Unitsrec);
                        cmd.Parameters.AddWithValue("@Id", id);
                        cmd.ExecuteNonQuery();
                    }
                }
                return Ok($"Record with ID {id} updated successfully.");
            }
            catch (SqlException ex)
            {
                return StatusCode(StatusCodes.Status500InternalServerError, ex.Message);
            }
        }

        [HttpDelete("{id}")]
        public IActionResult Delete(int id)
        {
            string sqlDataSource = _configuration.GetConnectionString("DefaultConnection");
            if (string.IsNullOrEmpty(sqlDataSource))
            {
                return StatusCode(StatusCodes.Status500InternalServerError, "DefaultConnection is
not defined in the configuration.");
            }

            try
```

```
            {
                using (SqlConnection conn = new SqlConnection(sqlDataSource))
                {
                    conn.Open();
                    string query = "DELETE FROM bb1 WHERE id = @Id";
                    using (SqlCommand cmd = new SqlCommand(query, conn))
                    {
                        cmd.Parameters.AddWithValue("@Id", id);
                        cmd.ExecuteNonQuery();
                    }
                }
                return Ok($"Record with ID {id} deleted successfully.");
            }
            catch (SqlException ex)
            {
                return StatusCode(StatusCodes.Status500InternalServerError, ex.Message);
            }
        }

        // Add other actions (POST, PUT, DELETE) here as needed
    }
}
```

Bb2Controller.cs:

```
using Microsoft.AspNetCore.Http;
using Microsoft.AspNetCore.Mvc;
using Microsoft.Extensions.Configuration;
using System;
using System.Data;
using System.Data.SqlClient;
using WebApplication2.Models;

namespace WebApplication2.Controllers
{
    [Route("api/[controller]")]
    [ApiController]
    public class Bb2Controller : ControllerBase
    {
        private readonly IConfiguration _configuration;

        public Bb2Controller(IConfiguration configuration)
        {
            _configuration = configuration;
        }

        [HttpGet]
        public IActionResult Get()
        {
            string sqlDataSource = _configuration.GetConnectionString("DefaultConnection");
            if (string.IsNullOrEmpty(sqlDataSource))
            {
                return StatusCode(StatusCodes.Status500InternalServerError, "DefaultConnection is
not defined in the configuration.");
            }

            try
            {
                using (SqlConnection conn = new SqlConnection(sqlDataSource))
                {
                    conn.Open();
```

```
                using (SqlCommand cmd = new SqlCommand("SELECT * FROM bb2", conn))
                {
                    SqlDataReader reader = cmd.ExecuteReader();
                    DataTable dataTable = new DataTable();
                    dataTable.Load(reader);

                    return Ok(dataTable);
                }
            }
        }
        catch (SqlException ex)
        {
            return StatusCode(StatusCodes.Status500InternalServerError, ex.Message);
        }
    }

    [HttpPost]
    public IActionResult Post(Bb2 bb2)
    {
        string sqlDataSource = _configuration.GetConnectionString("DefaultConnection");
        if (string.IsNullOrEmpty(sqlDataSource))
        {
            return StatusCode(StatusCodes.Status500InternalServerError, "DefaultConnection is
not defined in the configuration.");
        }

        try
        {
            using (SqlConnection conn = new SqlConnection(sqlDataSource))
            {
                conn.Open();
                string query = "INSERT INTO bb2 (bloodtype, totalcap) VALUES (@BloodType,
@TotalCap)";
                using (SqlCommand cmd = new SqlCommand(query, conn))
                {
                    cmd.Parameters.AddWithValue("@BloodType", bb2.BloodType);
                    cmd.Parameters.AddWithValue("@TotalCap", bb2.TotalCap);
                    cmd.ExecuteNonQuery();
                }
            }
            return Ok("Record inserted successfully.");
        }
        catch (SqlException ex)
        {
            return StatusCode(StatusCodes.Status500InternalServerError, ex.Message);
        }
    }

    [HttpPut("{id}")]
    public IActionResult Put(int id, Bb2 bb2)
    {
        string sqlDataSource = _configuration.GetConnectionString("DefaultConnection");
        if (string.IsNullOrEmpty(sqlDataSource))
        {
            return StatusCode(StatusCodes.Status500InternalServerError, "DefaultConnection is
not defined in the configuration.");
        }

        try
        {
            using (SqlConnection conn = new SqlConnection(sqlDataSource))
            {
```

```
                    conn.Open();
                    string query = "UPDATE bb2 SET totalcap = @TotalCap WHERE id = @Id";
                    using (SqlCommand cmd = new SqlCommand(query, conn))
                    {

                        cmd.Parameters.AddWithValue("@TotalCap", bb2.TotalCap);
                        cmd.Parameters.AddWithValue("@Id", id);
                        cmd.ExecuteNonQuery();
                    }
                }
                return Ok($"Record with ID {id} updated successfully.");
            }
            catch (SqlException ex)
            {
                return StatusCode(StatusCodes.Status500InternalServerError, ex.Message);
            }
        }

        [HttpDelete("{id}")]
        public IActionResult Delete(int id)
        {
            string sqlDataSource = _configuration.GetConnectionString("DefaultConnection");
            if (string.IsNullOrEmpty(sqlDataSource))
            {
                return StatusCode(StatusCodes.Status500InternalServerError, "DefaultConnection is
not defined in the configuration.");
            }

            try
            {
                using (SqlConnection conn = new SqlConnection(sqlDataSource))
                {
                    conn.Open();
                    string query = "DELETE FROM bb2 WHERE id = @Id";
                    using (SqlCommand cmd = new SqlCommand(query, conn))
                    {
                        cmd.Parameters.AddWithValue("@Id", id);
                        cmd.ExecuteNonQuery();
                    }
                }
                return Ok($"Record with ID {id} deleted successfully.");
            }
            catch (SqlException ex)
            {
                return StatusCode(StatusCodes.Status500InternalServerError, ex.Message);
            }
        }

        // Add other actions (POST, PUT, DELETE) here as needed
    }
}
```

DonorController.cs

```
using Microsoft.AspNetCore.Http;
using Microsoft.AspNetCore.Mvc;
using Microsoft.Extensions.Configuration;
using System;
using System.Data;
using System.Data.SqlClient;
using WebApplication2.Models;
```

```
namespace WebApplication2.Controllers
{
    [Route("api/[controller]")]
    [ApiController]
    public class DonorController : ControllerBase
    {
        private readonly IConfiguration _configuration;

        public DonorController(IConfiguration configuration)
        {
            _configuration = configuration;
        }

        [HttpGet]
        public IActionResult Get()
        {
            string sqlDataSource = _configuration.GetConnectionString("DefaultConnection");
            if (string.IsNullOrEmpty(sqlDataSource))
            {
                return StatusCode(StatusCodes.Status500InternalServerError, "DefaultConnection is
not defined in the configuration.");
            }

            try
            {
                using (SqlConnection conn = new SqlConnection(sqlDataSource))
                {
                    conn.Open();
                    using (SqlCommand cmd = new SqlCommand("SELECT * FROM donor", conn))
                    {
                        SqlDataReader reader = cmd.ExecuteReader();
                        DataTable dataTable = new DataTable();
                        dataTable.Load(reader);

                        return Ok(dataTable);
                    }
                }
            }
            catch (SqlException ex)
            {
                return StatusCode(StatusCodes.Status500InternalServerError, ex.Message);
            }
        }

        [HttpPost]
        public IActionResult Post(Donor donor)
        {
            string sqlDataSource = _configuration.GetConnectionString("DefaultConnection");
            if (string.IsNullOrEmpty(sqlDataSource))
            {
                return StatusCode(StatusCodes.Status500InternalServerError, "DefaultConnection is
not defined in the configuration.");
            }

            try
            {
                using (SqlConnection conn = new SqlConnection(sqlDataSource))
                {
                    conn.Open();
                    string query = "INSERT INTO donor (BID, DID, Units) VALUES (@BID, @DID,
@Units)";
```

```
                    using (SqlCommand cmd = new SqlCommand(query, conn))
                    {
                        cmd.Parameters.AddWithValue("@BID", donor.BID);
                        cmd.Parameters.AddWithValue("@DID", donor.DID);
                        cmd.Parameters.AddWithValue("@Units", donor.Units);
                        cmd.ExecuteNonQuery();
                    }
                }
                return Ok("Record inserted successfully.");
            }
            catch (SqlException ex)
            {
                return StatusCode(StatusCodes.Status500InternalServerError, ex.Message);
            }
        }

        [HttpPut("{id}")]
        public IActionResult Put(int id, Donor donor)
        {
            string sqlDataSource = _configuration.GetConnectionString("DefaultConnection");
            if (string.IsNullOrEmpty(sqlDataSource))
            {
                return StatusCode(StatusCodes.Status500InternalServerError, "DefaultConnection is
not defined in the configuration.");
            }

            try
            {
                using (SqlConnection conn = new SqlConnection(sqlDataSource))
                {
                    conn.Open();
                    string query = "UPDATE donor SET  Units=@Units WHERE id = @Id";
                    using (SqlCommand cmd = new SqlCommand(query, conn))
                    {



                        cmd.Parameters.AddWithValue("@Units", donor.Units);
                        cmd.Parameters.AddWithValue("@Id", id);
                        cmd.ExecuteNonQuery();
                    }
                }
                return Ok($"Record with ID {id} updated successfully.");
            }
            catch (SqlException ex)
            {
                return StatusCode(StatusCodes.Status500InternalServerError, ex.Message);
            }
        }

        [HttpDelete("{id}")]
        public IActionResult Delete(int id)
        {
            string sqlDataSource = _configuration.GetConnectionString("DefaultConnection");
            if (string.IsNullOrEmpty(sqlDataSource))
            {
                return StatusCode(StatusCodes.Status500InternalServerError, "DefaultConnection is
not defined in the configuration.");
            }

            try
            {
```

```
                using (SqlConnection conn = new SqlConnection(sqlDataSource))
                {
                    conn.Open();
                    string query = "DELETE FROM donor WHERE id = @Id";
                    using (SqlCommand cmd = new SqlCommand(query, conn))
                    {
                        cmd.Parameters.AddWithValue("@Id", id);
                        cmd.ExecuteNonQuery();
                    }
                }
                return Ok($"Record with ID {id} deleted successfully.");
            }
            catch (SqlException ex)
            {
                return StatusCode(StatusCodes.Status500InternalServerError, ex.Message);
            }
        }

        // Add other actions (POST, PUT, DELETE) here as needed
    }
}
```

Donor1Controller.cs:

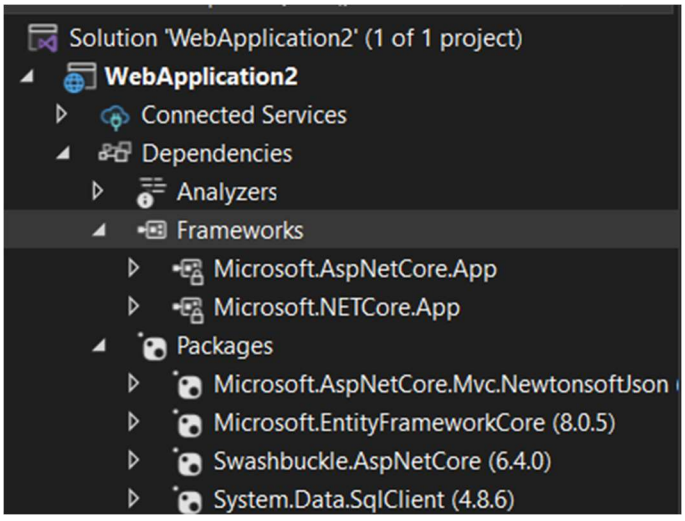
```
using Microsoft.AspNetCore.Http;
using Microsoft.AspNetCore.Mvc;
using Microsoft.Extensions.Configuration;
using System;
using System.Data;
using System.Data.SqlClient;
using WebApplication2.Models;
using System.Text.Json;
using System.Collections.Generic;

namespace WebApplication2.Controllers
{
    [Route("api/[controller]")]
    [ApiController]
    public class Donor1Controller : ControllerBase
    {
        private readonly IConfiguration _configuration;

        public Donor1Controller(IConfiguration configuration)
        {
            _configuration = configuration;
        }

        [HttpGet]
        public IActionResult Get()
        {
            string sqlDataSource = _configuration.GetConnectionString("DefaultConnection");
            if (string.IsNullOrEmpty(sqlDataSource))
            {
                return StatusCode(StatusCodes.Status500InternalServerError, "DefaultConnection is
not defined in the configuration.");
            }

            try
            {
                using (SqlConnection conn = new SqlConnection(sqlDataSource))
                {
```

```
                    conn.Open();
                    using (SqlCommand cmd = new SqlCommand("SELECT * FROM donor1", conn))
                    {
                        SqlDataReader reader = cmd.ExecuteReader();
                        DataTable dataTable = new DataTable();
                        dataTable.Load(reader);

                        return Ok(dataTable);
                    }
                }
            }
            catch (SqlException ex)
            {
                return StatusCode(StatusCodes.Status500InternalServerError, ex.Message);
            }
        }

        [HttpPost]
        public IActionResult Post(Donor1 donor1)
        {
            string sqlDataSource = _configuration.GetConnectionString("DefaultConnection");
            if (string.IsNullOrEmpty(sqlDataSource))
            {
                return StatusCode(StatusCodes.Status500InternalServerError, "DefaultConnection is
not defined in the configuration.");
            }

            try
            {
                using (SqlConnection conn = new SqlConnection(sqlDataSource))
                {
                    conn.Open();
                    string query = "INSERT INTO donor1 (DID, Name, Age, Gender, Contact,
Bloodtype) VALUES (@DID, @Name, @Age, @Gender, @Contact, @Bloodtype)";
                    using (SqlCommand cmd = new SqlCommand(query, conn))
                    {
                        cmd.Parameters.AddWithValue("@DID", donor1.DID);
                        cmd.Parameters.AddWithValue("@Name", donor1.Name);
                        cmd.Parameters.AddWithValue("@Age", donor1.Age);
                        cmd.Parameters.AddWithValue("@Gender", donor1.Gender);
                        cmd.Parameters.AddWithValue("@Contact", donor1.Contact);
                        cmd.Parameters.AddWithValue("@Bloodtype", donor1.Bloodtype);

                        cmd.ExecuteNonQuery();
                    }
                }
                return Ok("Record inserted successfully.");
            }
            catch (SqlException ex)
            {
                return StatusCode(StatusCodes.Status500InternalServerError, ex.Message);
            }
        }



        [HttpDelete("{id}")]
        public IActionResult Delete(int id)
        {
            string sqlDataSource = _configuration.GetConnectionString("DefaultConnection");
            if (string.IsNullOrEmpty(sqlDataSource))
            {
```

```
                return StatusCode(StatusCodes.Status500InternalServerError, "DefaultConnection is
not defined in the configuration.");
            }

            try
            {
                using (SqlConnection conn = new SqlConnection(sqlDataSource))
                {
                    conn.Open();
                    string query = "DELETE FROM donor1 WHERE id = @Id";
                    using (SqlCommand cmd = new SqlCommand(query, conn))
                    {
                        cmd.Parameters.AddWithValue("@Id", id);
                        cmd.ExecuteNonQuery();
                    }
                }
                return Ok($"Record with ID {id} deleted successfully.");
            }
            catch (SqlException ex)
            {
                return StatusCode(StatusCodes.Status500InternalServerError, ex.Message);
            }
        }

        // Add other actions (POST, PUT, DELETE) here as needed
    }
}
```

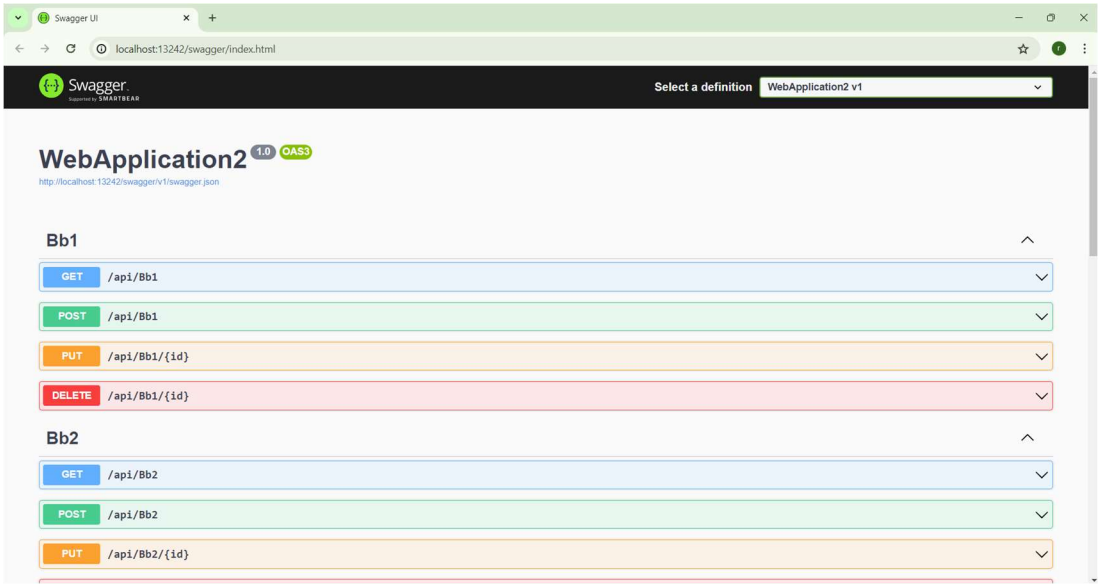
Frameworks and Packages used:

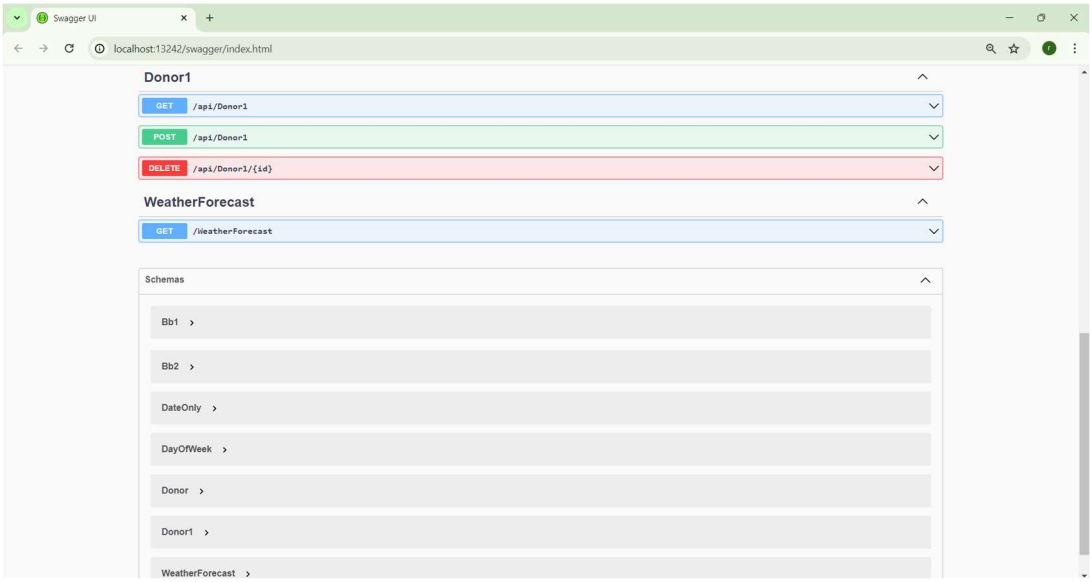
- Microsoft.Asp.NetCore.App
- Microsoft.NetCore.App



For the visual representation of the API and the documentation in the interactive format **Swagger** is used. (instead of postman)

The output of the created API is shown below.

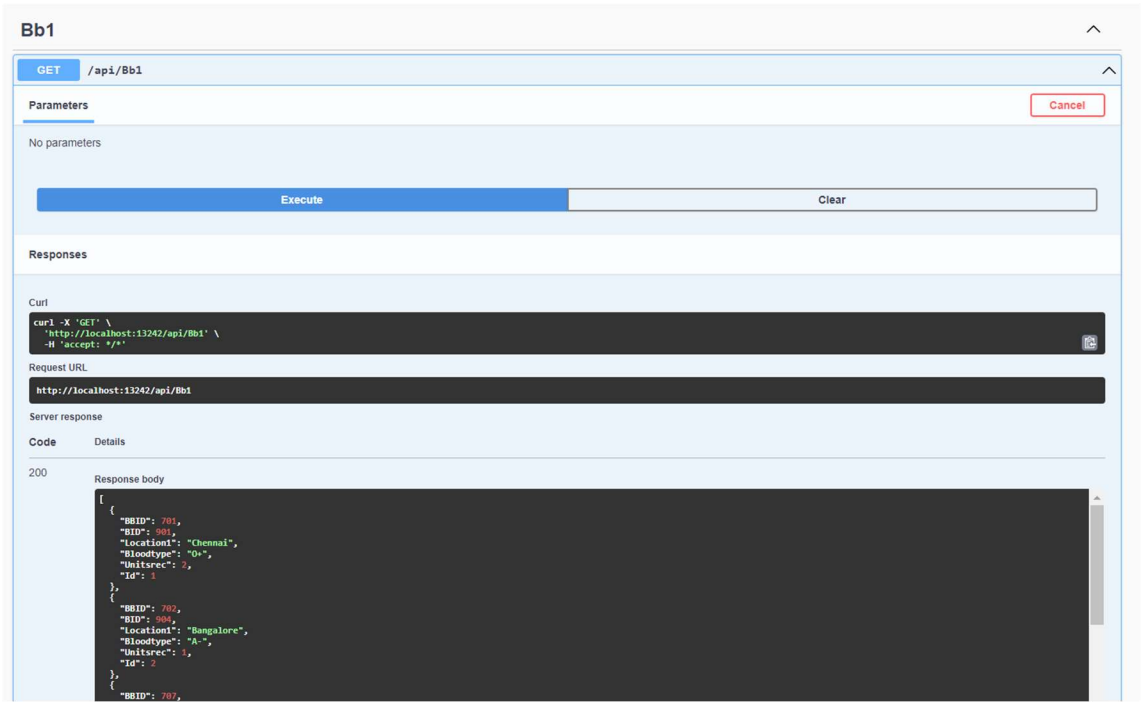


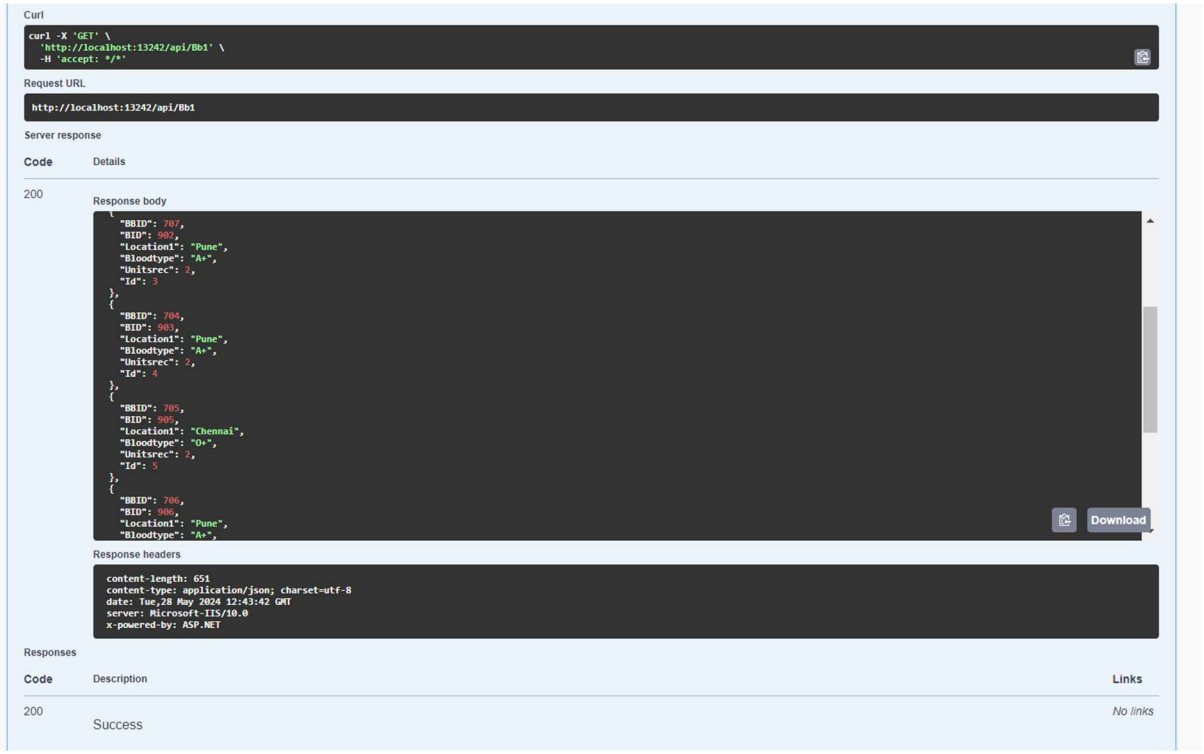


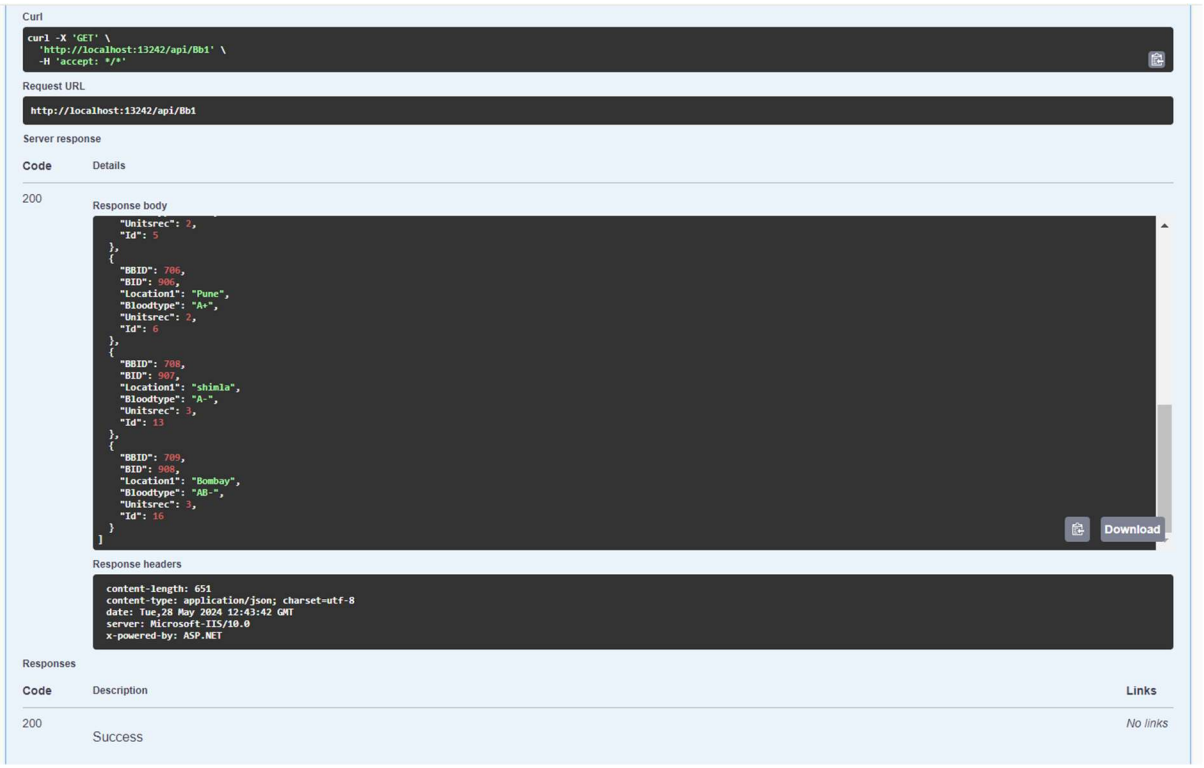
CRUD operation on the BB1 table

GET

The entries in the table in the backend as well as status of the table are displayed.

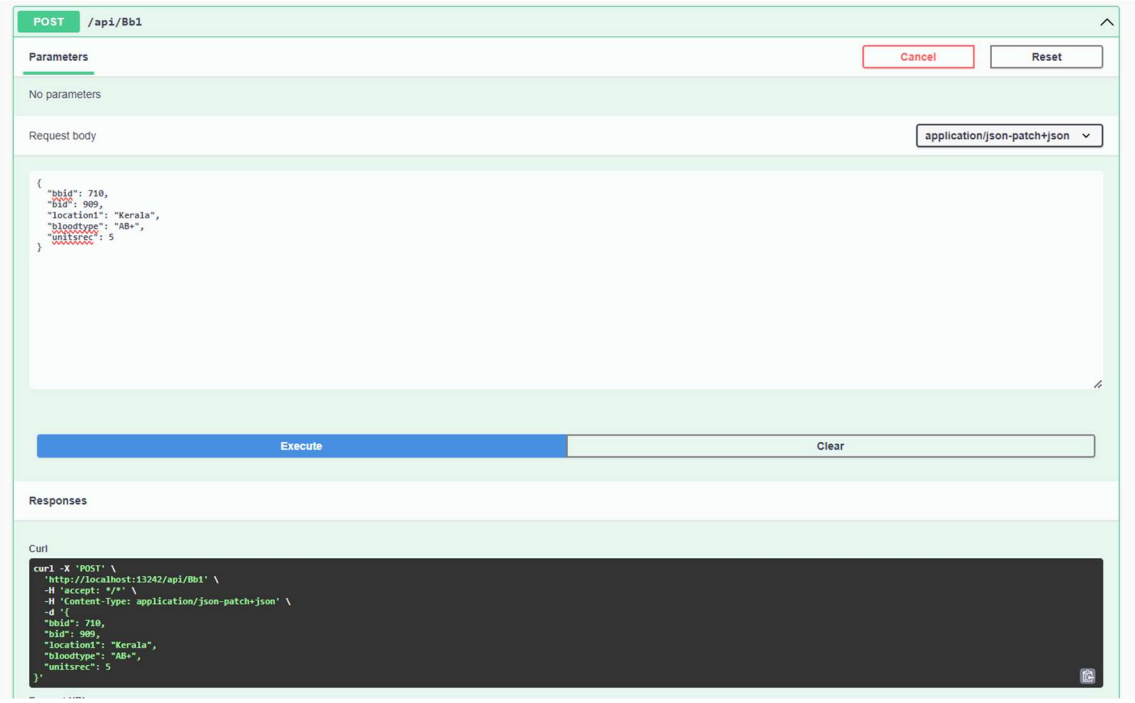


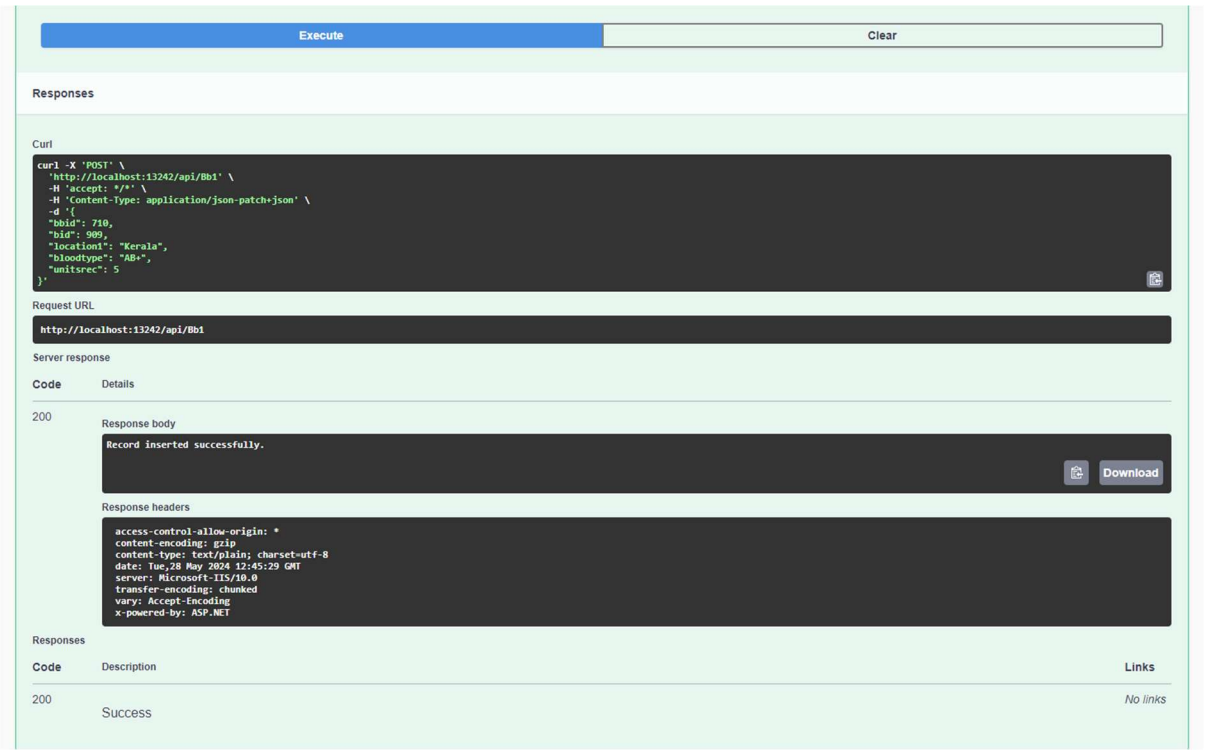




POST

The insertion is made with the help of POST http request such that the action reflects in the database table too.


POST
/api/Bb1
Parameters
Cancel
Reset
No parameters
Request body
application/json-patch+json
{
"bbid": 710,
"bid": 909,
"location1": "Kerala",
"bloodtype": "AB+",
"unitsrec": 5
}
Execute
Clear
Responses
Curl
curl -X 'POST' \
'http://localhost:13242/api/Bb1' \
-H 'accept: */*' \
-H 'Content-Type: application/json-patch+json' \
-d '{
"bbid": 710,
"bid": 909,
"location1": "Kerala",
"bloodtype": "AB+",
"unitsrec": 5
}'


Execute
Clear
Responses
Curl
curl -X 'POST' \
'http://localhost:13242/api/Bb1' \
-H 'accept: */*' \
-H 'Content-Type: application/json-patch+json' \
-d '{
"bbid": 710,
"bid": 909,
"location1": "Kerala",
"bloodtype": "AB+",
"unitsrec": 5
}'
Request URL
http://localhost:13242/api/Bb1
Server response
Code
Details
200
Response body
Record inserted successfully.
Download
Response headers
access-control-allow-origin: *
content-encoding: gzip
content-type: text/plain; charset=utf-8
date: Tue,28 May 2024 12:45:29 GMT
server: Microsoft-IIS/10.0
transfer-encoding: chunked
vary: Accept-Encoding
x-powered-by: ASP.NET
Responses
Code
Description
Links
200
Success
No links

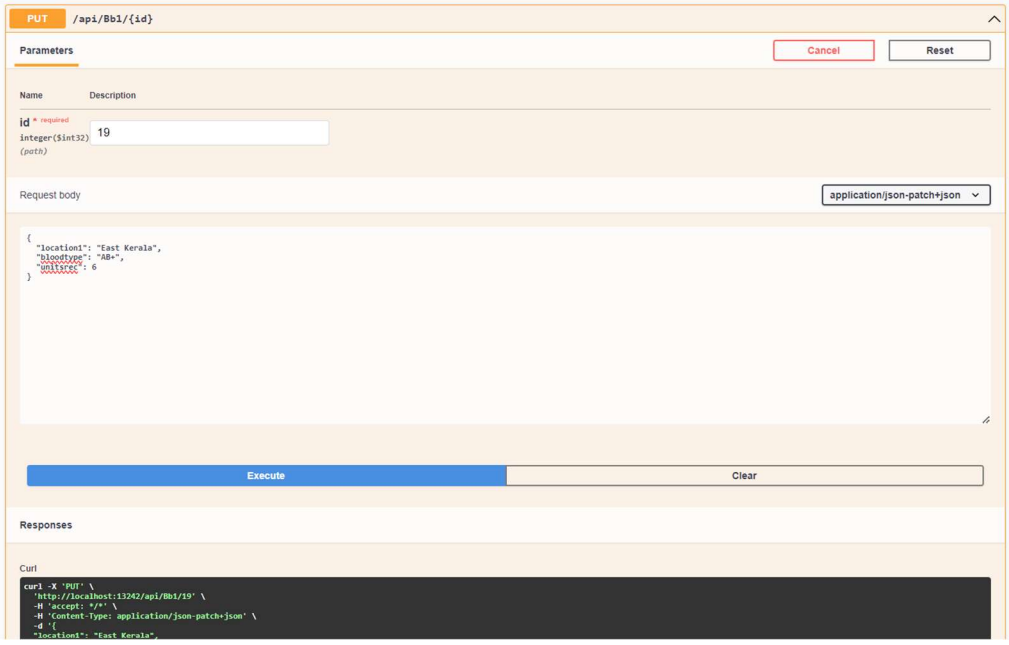
The updated table in the database is

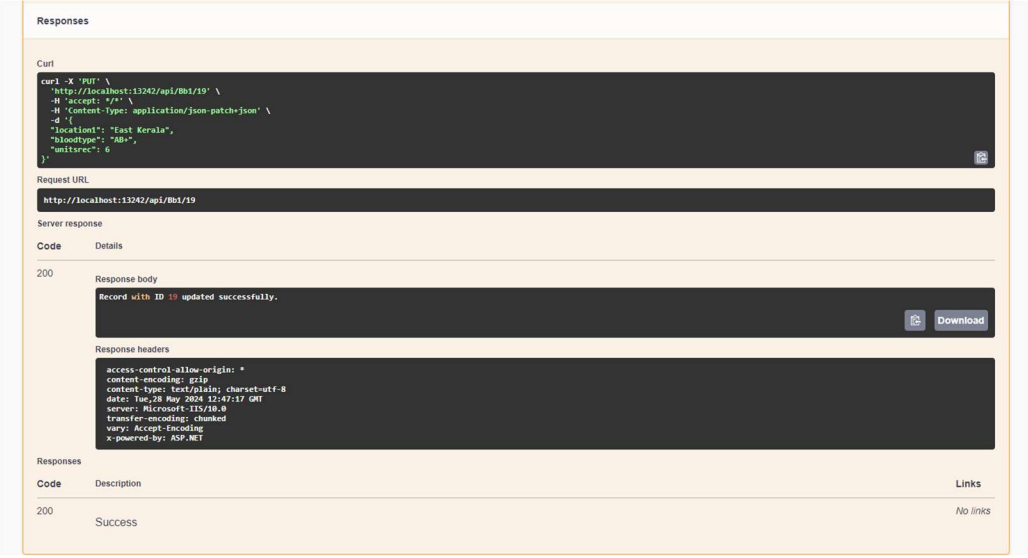
Results Messages

	BBID	BID	Location1	Bloodtype	Unitsrec	Id
1	701	901	Chennai	O+	2	1
2	702	904	Bangalore	A-	1	2
3	707	902	Pune	A+	2	3
4	704	903	Pune	A+	2	4
5	705	905	Chennai	O+	2	5
6	706	906	Pune	A+	2	6
7	708	907	shimla	A-	3	13
8	709	908	Bombay	AB-	3	16
9	710	909	Kerala	AB+	5	19

PUT

The updation is made with the help of PUT http request such that the physical table is also updated.





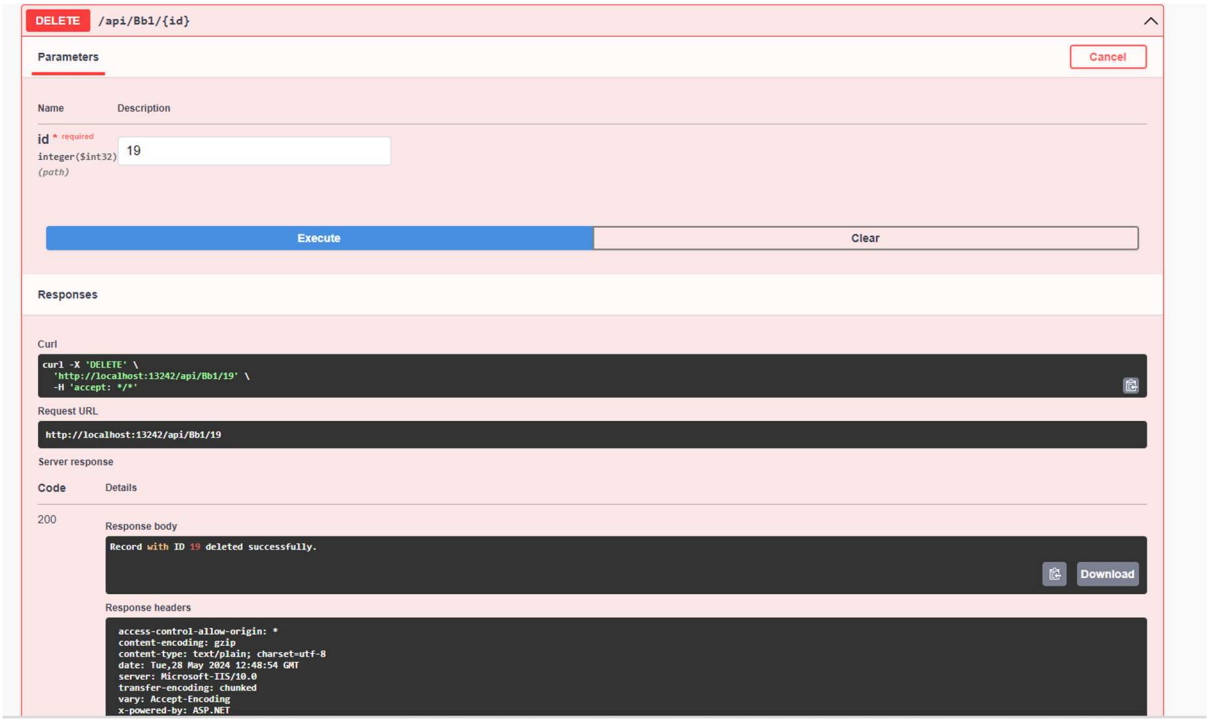
The updated table in the database is

Results Messages

	BBID	BID	Location1	Bloodtype	Unitsrec	Id
1	701	901	Chennai	O+	2	1
2	702	904	Bangalore	A-	1	2
3	707	902	Pune	A+	2	3
4	704	903	Pune	A+	2	4
5	705	905	Chennai	O+	2	5
6	706	906	Pune	A+	2	6
7	708	907	shimla	A-	3	13
8	709	908	Bombay	AB-	3	16
9	710	909	East Kerala	AB+	6	19

DELETE

The deletion is made with the help of DELETE http request such that the physical table is also reflected.



The updated table in the database is

Results Messages

	BBID	BID	Location1	Bloodtype	Unitsrec	Id
1	701	901	Chennai	O+	2	1
2	702	904	Bangalore	A-	1	2
3	707	902	Pune	A+	2	3
4	704	903	Pune	A+	2	4
5	705	905	Chennai	O+	2	5
6	706	906	Pune	A+	2	6
7	708	907	shimla	A-	3	13
8	709	908	Bombay	AB-	3	16

FRONTEND

For the frontend, react.js is used. *Index.js* is used to see the contents of the app. Components are injected into the div with id. Bootstrap is installed for styling and the header is added.

Index.html

```
<!DOCTYPE html>

<html lang="en">
  <head>
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/css/bootstrap.min.css"
rel="stylesheet">
    <meta charset="utf-8" />
    <link rel="icon" href="%PUBLIC_URL%/favicon.ico" />
    <meta name="viewport" content="width=device-width, initial-scale=1" />
    <meta name="theme-color" content="#000000" />
    <meta
      name="description"
      content="Web site created using create-react-app"
    />
    <link rel="apple-touch-icon" href="%PUBLIC_URL%/logo192.png" />
    <!--
      manifest.json provides metadata used when your web app is installed on a
      user's mobile device or desktop. See
https://developers.google.com/web/fundamentals/web-app-manifest/
    -->
    <link rel="manifest" href="%PUBLIC_URL%/manifest.json" />
    <!--
      Notice the use of %PUBLIC_URL% in the tags above.
      It will be replaced with the URL of the `public` folder during the build.
      Only files inside the `public` folder can be referenced from the HTML.

      Unlike "/favicon.ico" or "favicon.ico", "%PUBLIC_URL%/favicon.ico" will
      work correctly both with client-side routing and a non-root public URL.
      Learn how to configure a non-root public URL by running `npm run build`.
    -->
    <title>React App</title>
  </head>
  <body>
    <noscript>You need to enable JavaScript to run this app.</noscript>
    <div id="root"></div>
    <!--
      This HTML file is a template.
      If you open it directly in the browser, you will see an empty page.

      You can add webfonts, meta tags, or analytics to this file.
      The build step will place the bundled scripts into the <body> tag.

      To begin the development, run `npm start` or `yarn start`.
```

```
      To create a production bundle, use `npm run build` or `yarn build`.
    -->
    <script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/js/bootstrap.bundle.min.js"></script>
  </body>
</html>
```

App.js

App.js is the root component which contains the contents that would be displayed on the screen. It includes pages that connects to each table separately to display data and to perform CRUD operations.

```
import logo from './logo.svg';
import './App.css';
import {Home} from './Home';
import {BB1} from './BB1';
import {BB2} from './BB2';
import {Donor} from './Donor';
import {Donor1} from './Donor1';
import { BrowserRouter, Route, Switch, NavLink } from 'react-router-dom';



function App() {
  return (
    <BrowserRouter>
    <div className="App-container">
      <h3 className="d-flex justify-content-center m-3">
        Blood Donation Management System
      </h3>

      <nav className="navbar navbar-expand-sm bg-light navbar-dark">
        <ul className="navbar-nav">
          <li className="nav-item- m-1">
            <NavLink className="btn btn-light btn-outline-primary" to="/home">
              Home
            </NavLink>
          </li>
          <li className="nav-item- m-1">
            <NavLink className="btn btn-light btn-outline-primary" to="/bb1">
              BB1
            </NavLink>
          </li>
          <li className="nav-item- m-1">
            <NavLink className="btn btn-light btn-outline-primary" to="/bb2">
              BB2
```

```
            </NavLink>
          </li>
          <li className="nav-item- m-1">
            <NavLink className="btn btn-light btn-outline-primary" to="/donor">
              Donor
            </NavLink>
          </li>
          <li className="nav-item- m-1">
            <NavLink className="btn btn-light btn-outline-primary" to="/donor1">
              Donor1
            </NavLink>
          </li>
        </ul>

      </nav>

      <Switch>
          <Route path='/home' component={Home}/>
          <Route path='/bb1' component={BB1}/>
          <Route path='/bb2' component={BB2}/>
          <Route path='/donor' component={Donor}/>
          <Route path='/donor1' component={Donor1}/>
      </Switch>

    </div>
    </BrowserRouter>
  );
}

export default App;
```

Variables.js

variable.js is used such that it is connected to the API using the port number *13242* specific to the localhost here.

```
export const variables={
    API_URL:"http://localhost:13242/api/"
}
```

Bb1.js

```
import React, {Component} from 'react';
import {variables} from './Variables';

export class BB1 extends Component{
    constructor(props){
        super(props);
            this.state={
                bb1:[],
                modalTitle:"",

                BBID:"",
                BID:"",
                Location1:"",
                Bloodtype:"",
                Unitsrec:"",
                Id:0


        };
    }

    refreshList() {
        fetch(variables.API_URL + 'bb1')
            .then(response => {
                if (!response.ok) {
                    throw new Error('Network response was not ok ' + response.statusText);
                }
                return response.json();
            })
            .then(data => {
                this.setState({ bb1: data });
            })
            .catch(error => {
                console.error('There was a problem with the fetch operation:', error);
            });
    }

    componentDidMount(){
        this.refreshList();
    }
    changeBBID = (e) => {
        this.setState({ BBID: e.target.value });
    }
```

```
changeBID = (e) => {
    this.setState({ BID: e.target.value });
}
changeLocation = (e) => {
    this.setState({ Location1: e.target.value });
}

changeBloodtype = (e) => {
    this.setState({ Bloodtype: e.target.value });
}

changeUnitsrec=(e)=>{
    this.setState({Unitsrec:e.target.value})
}

addClick = () => {
    this.setState({
        modalTitle: "Add new BBID",
        BBID: "", // Set BBID to an empty string to allow user input
        BID: "",
        Location1: "",
        Bloodtype: "",
        Unitsrec: "",
        Id:0
    });
}

editClick = (bba1) => {
    this.setState({
        modalTitle: "Edit BBID",
        Id: bba1.Id,
        BBID: bba1.BBID,
        BID: bba1.BID,
        Bloodtype: bba1.Bloodtype,
        Unitsrec: bba1.Unitsrec,
        Location1: bba1.Location1
    });
}

createClick(){
    fetch(variables.API_URL+'bb1',{
        method:'POST',
        headers:{
            'Accept':'application/json',
            'Content-type': 'application/json'
        },
        body:JSON.stringify({
```

```
                BBID:this.state.BBID,
                BID:this.state.BID,
                Location1:this.state.Location1,
                Bloodtype:this.state.Bloodtype,
                Unitsrec:this.state.Unitsrec

            })
        })
        .then(res => {
            if (!res.ok) {
                throw new Error('Failed to create BBID');
            }
            return res.json();
        })
        .then(result => {
            alert('New BBID added successfully');
            this.refreshList();
        })
        .catch(error => {
            alert('Failed to create BBID: ' + error.message);
        });
    }


    updateClick(){
        fetch(variables.API_URL + 'bb1/' + this.state.Id, {
            method: 'PUT',
            headers: {
                'Accept': 'application/json',
                'Content-type': 'application/json'
            },
            body: JSON.stringify({
                BBID: this.state.BBID,
                BID: this.state.BID,
                Location1: this.state.Location1,
                Bloodtype: this.state.Bloodtype,
                Unitsrec: this.state.Unitsrec
            })
        })
        .then(res => {
            if (!res.ok) {
                throw new Error('Failed to update BBID');
            }
            return res.json();
        })
        .then(result => {
            alert('BBID updated successfully');
            this.refreshList();
```

```
        })
        .catch(error => {
            alert('Failed to update BBID: ' + error.message);
        });
    }

    render(){
        const {
            bb1,
            modalTitle,

            BBID,
            BID,
            Location1,
            Bloodtype,
            Unitsrec,
            Id

        }=this.state;

        return(
<div>
    <button type="button"
    className="btn btn-primary m-2 float-end"
    data-bs-toggle="modal"
    data-bs-target="#exampleModal"
    onClick={()=>this.addClick}>
        Add new BBID
    </button>
    <table className="table table-striped">
        <thead>
            <tr>
                <th>
                    BBID
                </th>
                <th>
                    BID
                </th>
                <th>
                    Location
                </th>
                <th>
                    Bloodtype
                </th>
                <th>
                    Units recieved
                </th>
                <th>
```

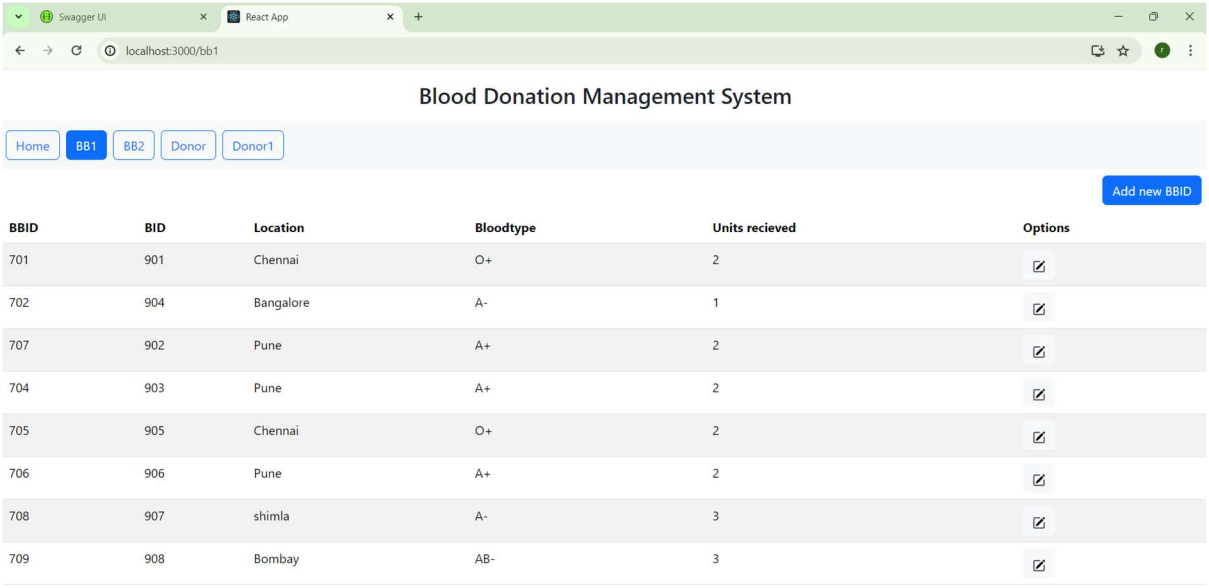
```
                    Options
                </th>
            </tr>
        </thead>
        <tbody>
            {bb1.map(bba1=>
                <tr key={bba1.BBID}>
                    <td>{bba1.BBID}</td>
                    <td>{bba1.BID}</td>
                    <td>{bba1.Location1}</td>
                    <td>{bba1.Bloodtype}</td>
                    <td>{bba1.Unitsrec}</td>

                    <td>
                        <button type="button"
                        className="btn btn-light mr-1"
                        data-bs-toggle="modal"
                        data-bs-target="#exampleModal"
                        onClick={()=>this.editClick(bba1)}>
                            <svg xmlns="http://www.w3.org/2000/svg" width="16" height="16"
fill="currentColor" className="bi bi-pencil-square" viewBox="0 0 16 16">
                            <path d="M15.502 1.94a.5.5 0 0 1 0 .706L14.459 3.69l-2-
2L13.502.646a.5.5 0 0 1 .707 0l1.293 1.293zm-1.75 2.456-2-2L4.939 9.21a.5.5 0 0 0-.121.196l-
.805 2.414a.25.25 0 0 0 .316.316l2.414-.805a.5.5 0 0 0 .196-.12l6.813-6.814z"/>
                            <path fillRule="evenodd" d="M1 13.5A1.5 1.5 0 0 0 2.5 15h11a1.5
1.5 0 0 0 1.5-1.5v-6a.5.5 0 0 0-1 0v6a.5.5 0 0 1-.5.5h-11a.5.5 0 0 1-.5-.5v-11a.5.5 0 0 1 .5-
.5H9a.5.5 0 0 0 0-1H2.5A1.5 1.5 0 0 0 1 2.5z"/>
                            </svg>
                        </button>


                    </td>
                </tr>
            )}

        </tbody>
    </table>

<div className="modal fade" id="exampleModal" tabIndex="-1" aria-hidden="true">
    <div className="modal-dialog modal -lg modal-dialog-centered">
        <div className="modal-content">
            <div className="modal-header">
                <h5 className="modal-title">{modalTitle}</h5>
                <button type="button" className="btn-close" data-bs-dismiss="modal" aria-
label="Close">

                </button>
            </div>
```

```
            <div className="modal-body">

                <div className="input-group mb-3">
                    <span className="input-group-text">BBID</span>
                    <input type="text" className="form-control" value={BBID} onChange={(e) =>
this.setState({ BBID: e.target.value })} />
                </div>
                <div className="input-group mb-3">
                    <span className="input-group-text">BID</span>
                    <input type="text" className="form-control" value={BID} onChange={(e) =>
this.setState({ BID: e.target.value })} />
                </div>
                <div className="input-group mb-3">
                    <span className="input-group-text">Bloodtype</span>
                    <input type="text" className="form-control" value={Bloodtype}
onChange={(e) => this.setState({ Bloodtype: e.target.value })} />
                </div>
                <div className="input-group mb-3">
                    <span className="input-group-text">Location</span>
                    <input type="text" className="form-control" value={Location1}
onChange={(e) => this.setState({ Location1: e.target.value })} />
                </div>
                <div className="input-group mb-3">
                    <span className="input-group-text">Unitsrec</span>
                    <input type="text" className="form-control" value={Unitsrec}
onChange={(e) => this.setState({ Unitsrec: e.target.value })} />
                </div>

                {Id===0?
                <button type="button"
                className="btn btn-primary float-start"
                onClick={()=>this.createClick()}
                >Create</button>
                :null}

                {Id!==0?
                <button type="button"
                className="btn btn-primary float-start"
                onClick={()=>this.updateClick()}
                >Update</button>
                :null}

            </div>
        </div>

    </div>
</div>
</div>
```

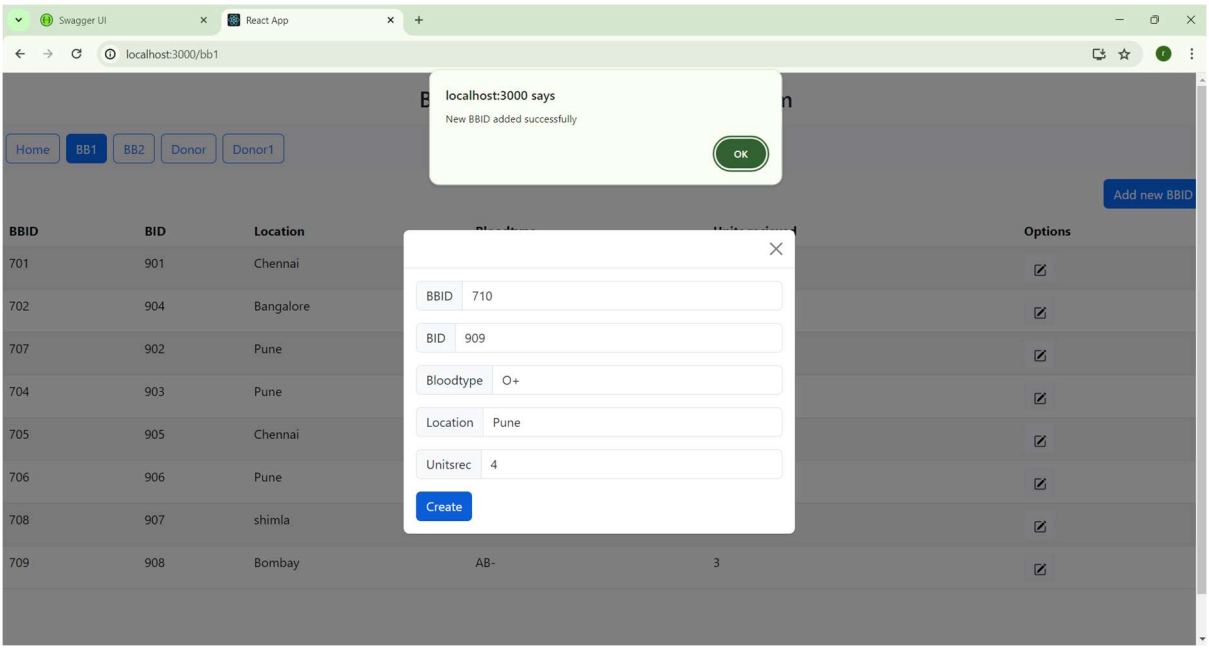
```
        )
    }
}
```

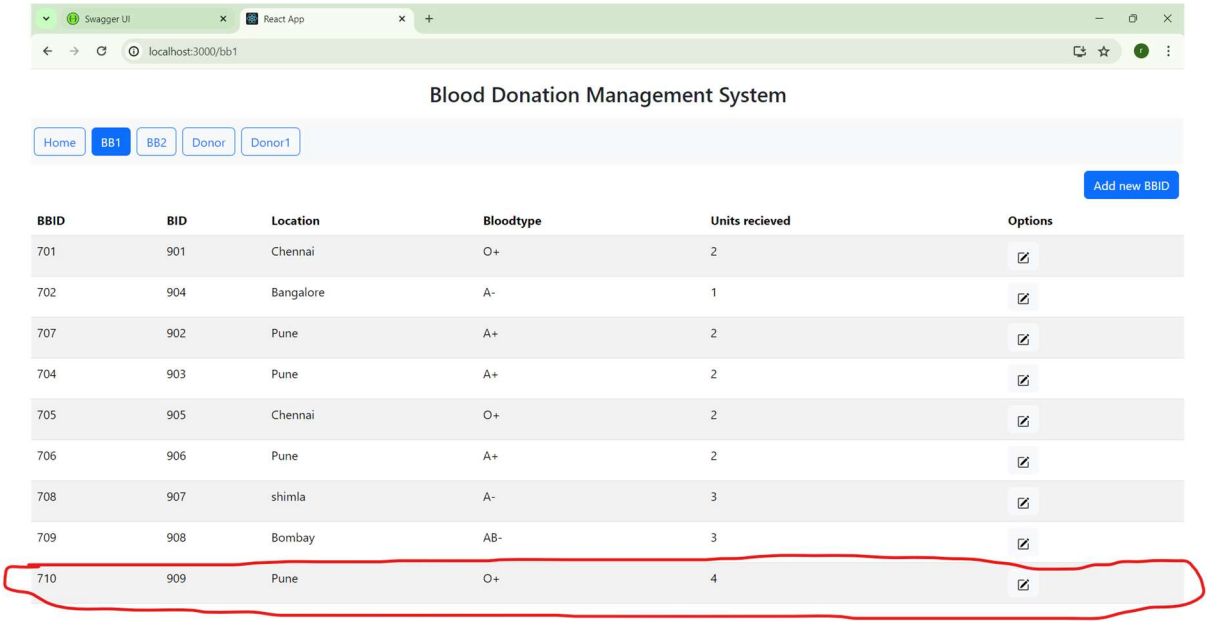
FRONTEND OUTPUT

Displaying data

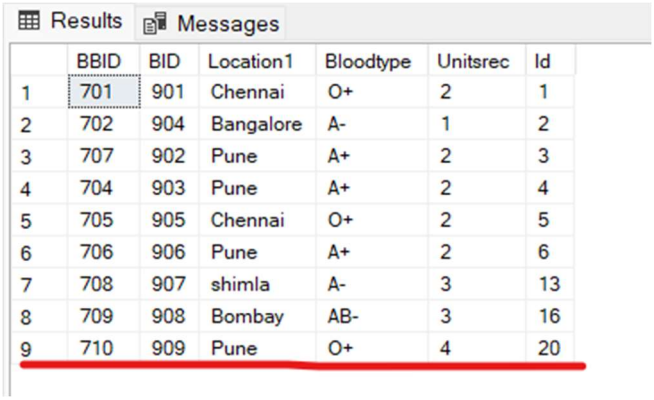


Insertion of new entry

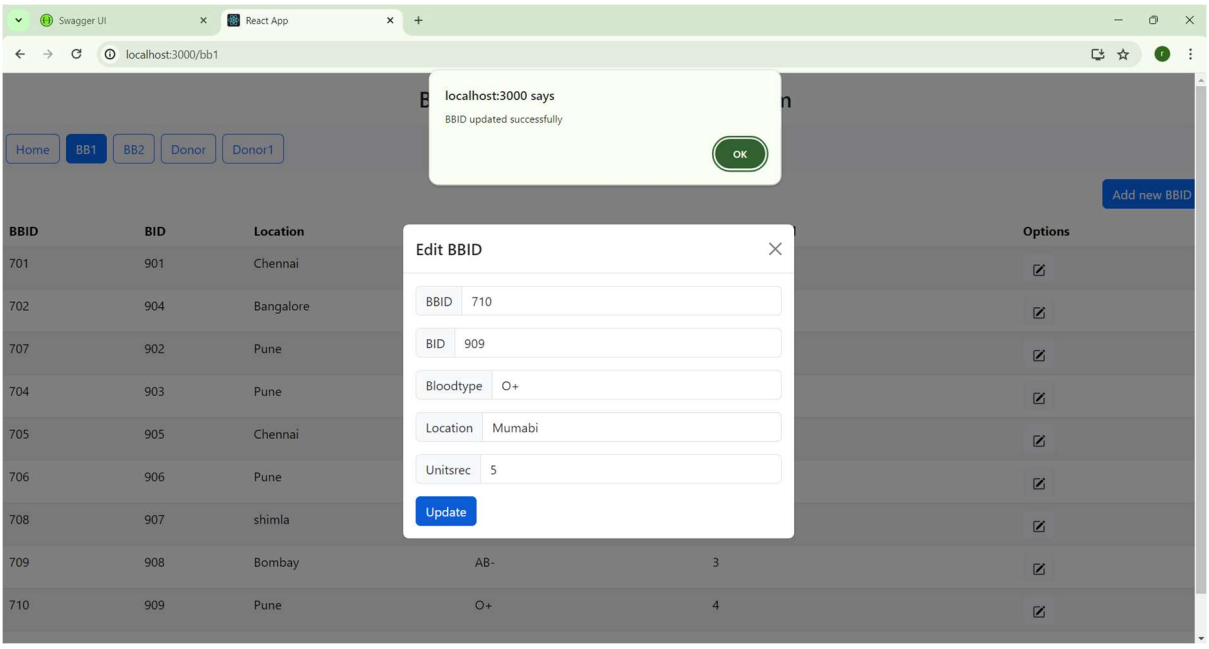


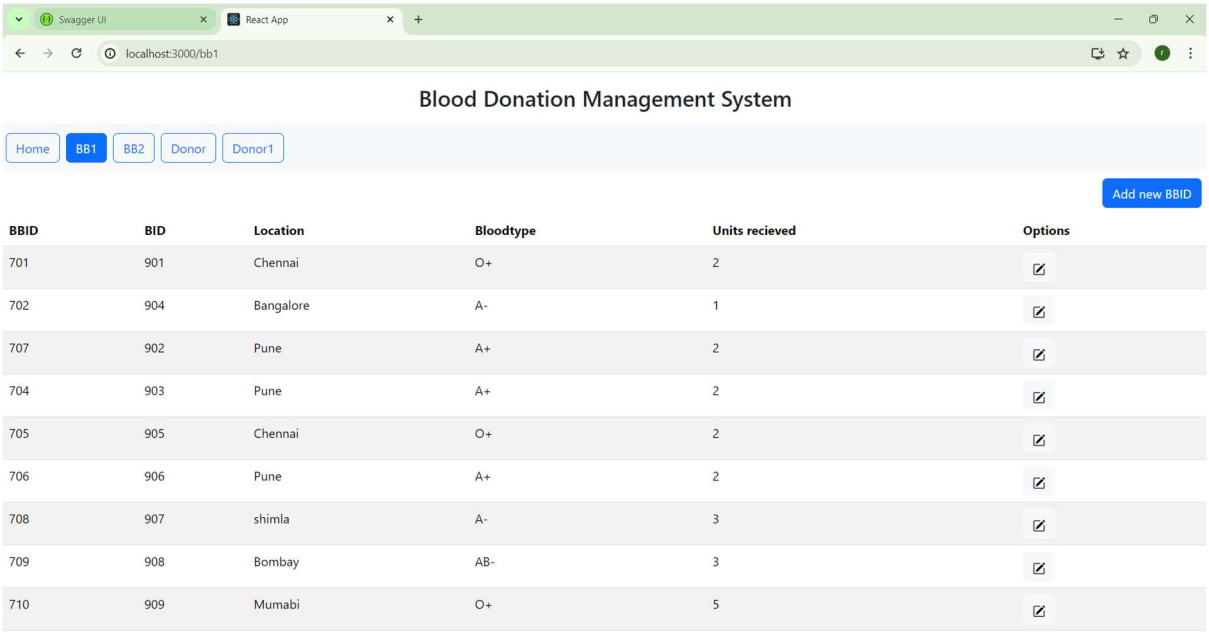


The updated table in database



Updation of the records





BBID	BID	Location	Bloodtype	Units recieved	Options
701	901	Chennai	O+	2	
702	904	Bangalore	A-	1	
707	902	Pune	A+	2	
704	903	Pune	A+	2	
705	905	Chennai	O+	2	
706	906	Pune	A+	2	
708	907	shimla	A-	3	
709	908	Bombay	AB-	3	
710	909	Mumabi	O+	5	

The updated table in the database

Results Messages

	BBID	BID	Location1	Bloodtype	Unitsrec	Id
1	701	901	Chennai	O+	2	1
2	702	904	Bangalore	A-	1	2
3	707	902	Pune	A+	2	3
4	704	903	Pune	A+	2	4
5	705	905	Chennai	O+	2	5
6	706	906	Pune	A+	2	6
7	708	907	shimla	A-	3	13
8	709	908	Bombay	AB-	3	16
9	710	909	Mumabi	O+	5	20

CONCLUSION

- Microsoft SQL Server Management Studio is used to create the database and its respective tables needed for the Blood donation management system. Specific DML and DDL commands were performed. Retrieval queries and triggers were implemented.
- A full-stack web application is developed using ASP.NET Core Web API, Microsoft SQL Server and React.js. The backend driven by ASP.NET Core Web API is integrated with the database, and the frontend is developed using react.js for dynamic, responsive user interface and is made to communicate with backend to perform CRUD operations on the data.