**Dot Net Phase 4 – Custom Support Logger**

* You need to create an ASP.Net MVC application within the Docker container and push it to GitHub by cloning the empty repository.
* You need to create Database Azure SQL Database to store data.
* After implementing all the functionalities, you must push the project to GitHub and then build a project using the Jenkins server by pulling the project from the GitHub repository.

1. **Project Creation:**

Create a repository on GitHub and clone the empty repository on the local machine. Inside the local repository, create the below projects using Visual Studio:

a. Create a class library project (DAL)

b. Create a class library project (DALTest)

c. Create ASP.Net Web MVC application project with Docker Support

**2. Functionalities:**

**A.** Create an SQL Server Database on Azure with the structure given below:

Table: **UserInfo**

|  |  |  |
| --- | --- | --- |
| **Field Name** | **Type** | **Constraints** |
| UserId | int | Primary Key |
| Email | nvarchar(100) |  |
| Password | nvarchar(20) |  |

Table: **CustLogInfo**

|  |  |  |
| --- | --- | --- |
| **Property Name** | **Type** | **Constraints** |
| LogId | int | Primary Key |
| CustEmail | nvarchar(100) |  |
| CustName | nvarchar(50) |  |
| LogStatus | nvarchar(50) |  |
| UserId | int | Foreign key |
| Description | nvarchar(50) |  |

1. **DAL:** In this layer, add the entity data model by selecting SQL Server database which is created on Azure and add the below functionalities using the data repository pattern
   1. Add DAL class and write a function to validate the user (customer support executive) from the UserInfo table using entity framework
   2. Add one more DAL class to save complaint log information to CustLogInfo table using the entity framework

1. **DALTest**: In this layer, test functionalities written in DAL using NUnit and Moq framework, such as UserInfo and CustLogInfo functionalities
2. **CustomerSupportLogger:** This is an MVC application to consume functionalities you have written in DAL:
   1. Develop a login page to validate user (customer support executive) as shown in the output
   2. If the user is valid, then develop a page to add customer complaints as shown in the output
   3. Debug this application on the Docker container as shown in the output

3. Push the entire project over a GitHub repository using the Visual Studio Git extension

4. After pushing a project to GitHub, create a job in Jenkins to build a project which has been pushed over a GitHub

5. Create a **FreeStyle** project in Jenkins, and configure it as mentioned in the next few steps

6. Configure Git **Source Code Management,** add GitHub project URL, and set branch as **Main**

7. To Trigger a build, select **Poll SCM** and schedule the build in such a way that the project triggers the build process after each hour

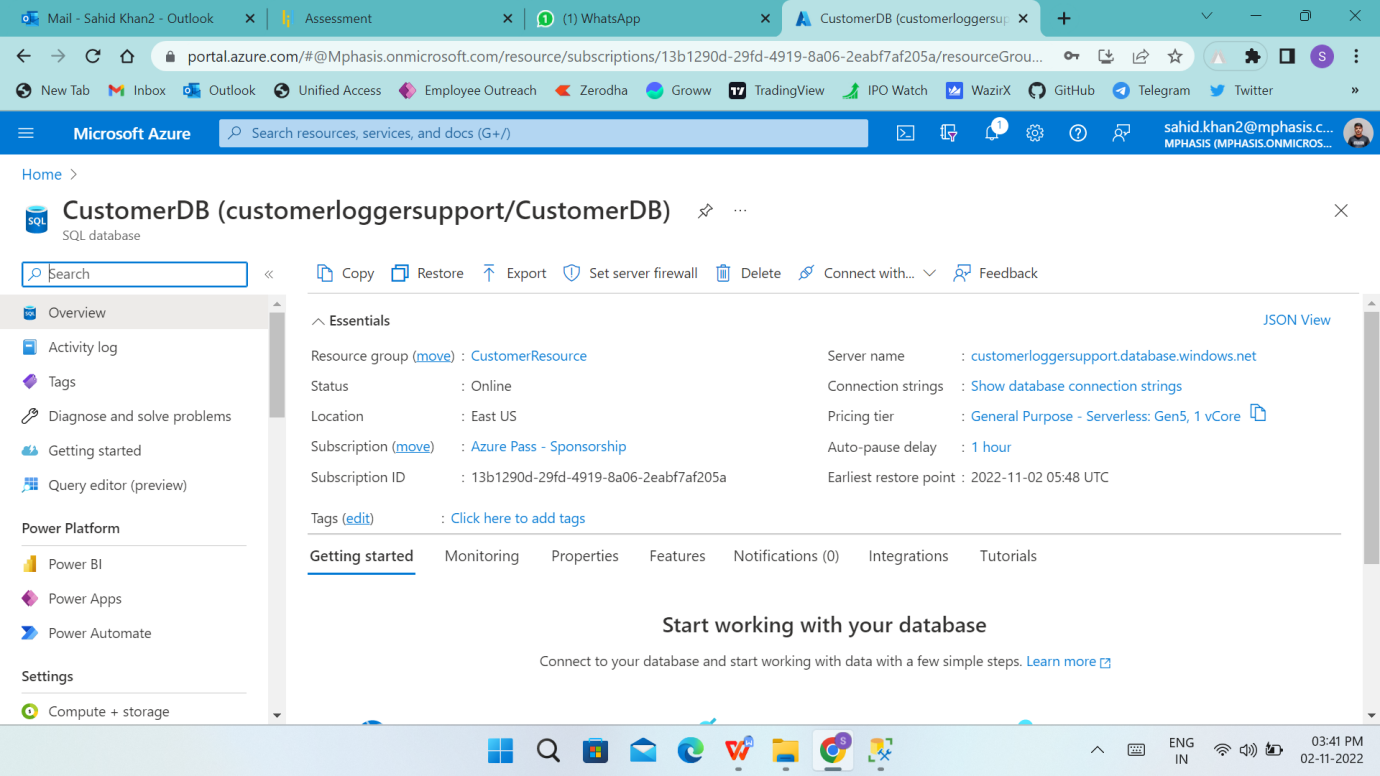
8. Set **Build** as **(Build a Visual Studio project or Solution using MSBuild)**

9. Select MSBuild version installed in Jenkins, and write .sln file name with the relative path, which exists inside the GitHub repository

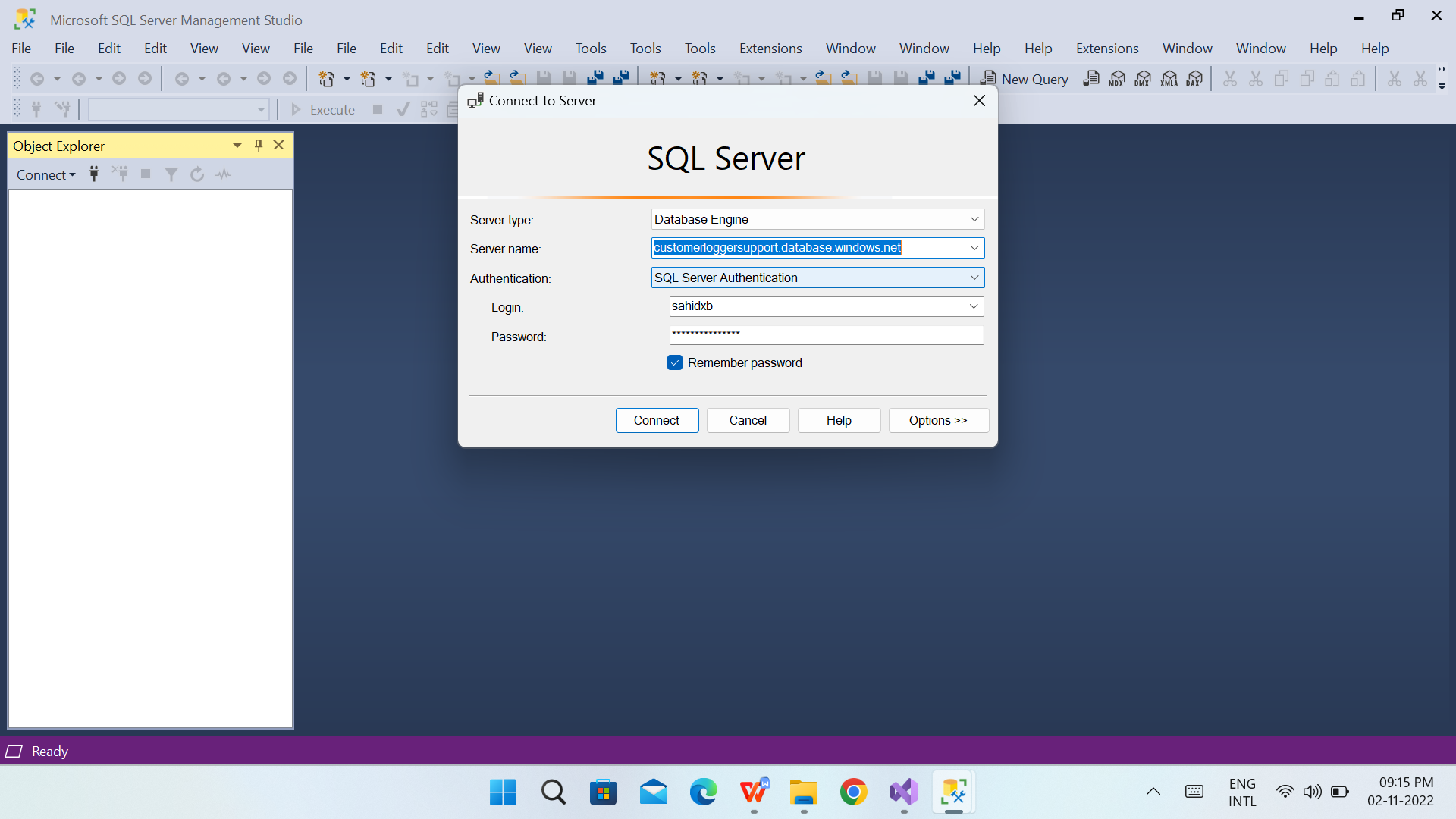
10. Build a Project in Jenkins

**Steps:**

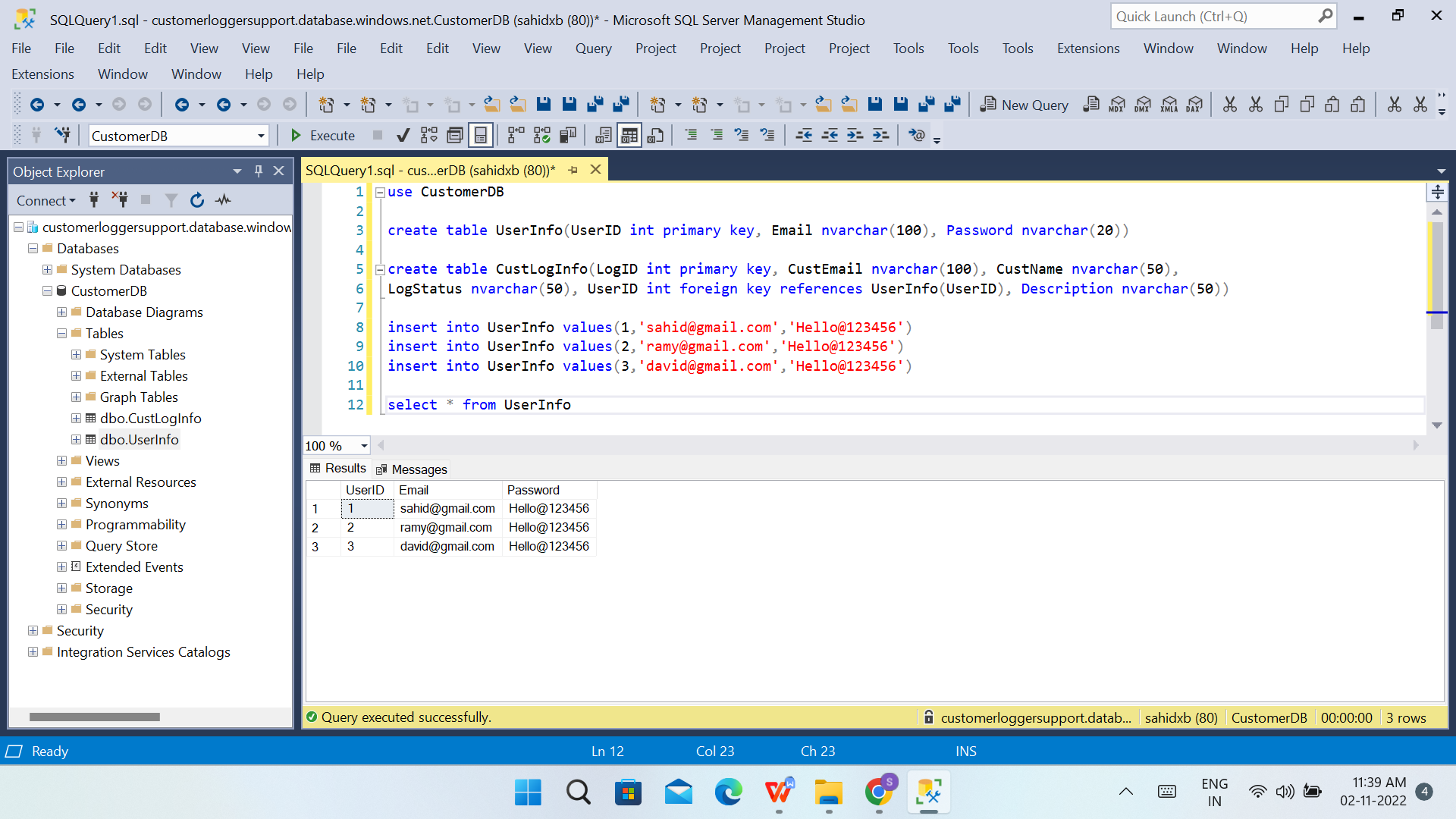
1. **First create a SQL DB and Server in Azure Portal.**



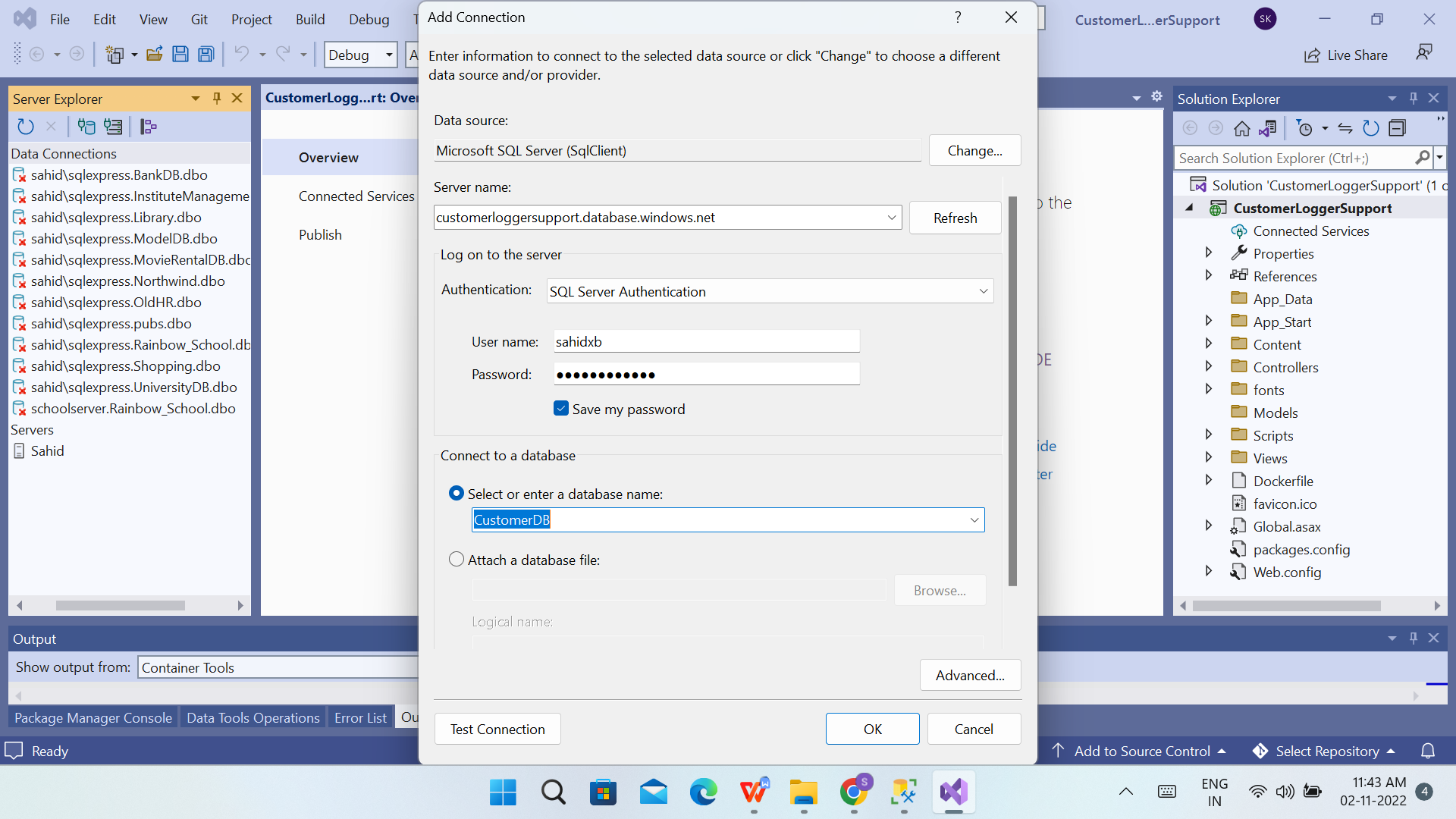
1. **Connect to SSMS with server name, user name and passsword..**



1. **Now create 2 table n insert data in 1st table to validate.**



1. **Now Open Visual Studio, Create project(ASP.NET web app .NET Framework) with docker support and then Connect database engine.**



1. **Now Do DB first approach in DAL Library(with installing Entity framework).**

**To know more check github…**

1. **Now create ValidateUser.cs class in DAL.**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace CustomerDAL

{

public class ValidateUsers

{

CustomerDBEntities Context = null;

public ValidateUsers()

{

Context = new CustomerDBEntities();

}

public bool ValidateUser(int userid, string password)

{

bool ans = false;

var find = Context.UserInfoes.ToList();

var find2 = find.Find(x => x.UserID == userid);

if (find2 != null)

{

if (find2.Password == password)

{

ans = true;

}

}

return ans;

}

public bool Insert(CustLogInfo c)

{

bool ans = true;

try

{

Context.CustLogInfoes.Add(c);

Context.SaveChanges();

return ans;

}

catch (Exception)

{

ans = false;

return ans;

}

}

}

**In App.config**

<connectionStrings>

<add name="CustomerDBEntities" connectionString="metadata=res://\*/Model1.csdl|res://\*/Model1.ssdl|res://\*/Model1.msl;provider=System.Data.SqlClient;provider connection string=&quot;data source=customerloggersupport.database.windows.net;initial catalog=CustomerDB;persist security info=True;user id=sahidxb;password=Hello@123456;MultipleActiveResultSets=True;App=EntityFramework&quot;" providerName="System.Data.EntityClient" />

</connectionStrings>

1. **Now create 2 Model, User n CustLogInfo..**

**user**

namespace CustomerLoggerSupport.Models

{

public class UserModel

{

public int UserID { get; set; }

public string Email { get; set; }

public string Password { get; set; }

}

}

**CustLogInfo**

namespace CustomerLoggerSupport.Models

{

public class CustLogInfoModel

{

public int LogID{ get; set; }

public string CustEmail { get; set; }

public string CustName { get; set; }

public string LogStatus { get; set; }

public int UserID { get; set; }

public string Description { get; set; }

}

}

1. **Now create a controller - Customer**

using CustomerDAL;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

using System.Web.Mvc;

namespace CustomerLoggerSupport.Controllers

{

public class CustomerController : Controller

{

ValidateUsers user = null;

public CustomerController()

{

user = new ValidateUsers();

}

public ActionResult Index()

{

return View();

}

[HttpPost]

public ActionResult Index(FormCollection collection)

{

bool valid = user.ValidateUser(Convert.ToInt32(Request["userid"]),Request["password"]);

if (valid)

{

return RedirectToAction("CustomerLogger");

}

else

{

return RedirectToAction("Index");

}

}

public ActionResult CustomerLogger()

{

return View();

}

[HttpPost]

public ActionResult CustomerLogger(FormCollection collection)

{

CustLogInfo c = new CustLogInfo();

c.LogID = Int32.Parse(Request["LogID"]);

c.CustEmail = Request["CustEmail"];

c.CustName = Request["CustName"];

c.LogStatus = Request["LogStatus"];

c.UserID = Int32.Parse(Request["UserId"]);

c.Description = Request["Description"];

user.Insert(c);

ViewBag.Message = "Complaint is Registered Succesfully";

return View();

//return RedirectToAction("Index");

}

}

}

1. **Now add views from Customer Controller**

**Index.cshtml**

@{

Layout = null;

}

<!DOCTYPE html>

<html>

<head>

</head>

<body>

<fieldset id="UserInformation">

<legend>Customer Support Executive Login Page</legend>

<div class="card my-5">

@using (Html.BeginForm("Index", "Customer",

new { ReturnUrl = ViewBag.ReturnUrl },

FormMethod.Post, new { @class = "card-body cardbody-color p-lg-5" }))

{

@Html.AntiForgeryToken()

<div class="mb-3">

User ID:<input type="text" class="form-control" id="userid" name="userid" placeholder="Enter User ID">

</div>

<div class="mb-3">

Password:<input type="password" class="form-control" id="password" name="password" placeholder="Enter Password">

</div>

<div class="text-center"><button type="submit" class="btn btn-color px-5 mb-5 w-100">Login</button>

</div>

}

</div>

</fieldset>

</body>

</html>

**CustomerLogger.cshtml**

@model CustomerLoggerSupport.Models.CustLogInfoModel

@{

Layout = null;

}

<!DOCTYPE html>

<html>

<head>

<meta name="viewport" content="width=device-width" />

<title>CustomerLogger</title>

</head>

<body>

<fieldset id="CustomersInfo">

<legend>Add Customer Complaints</legend>

@using (Html.BeginForm())

{

@Html.AntiForgeryToken()

<div class="form-horizontal">

<div style="color:lawngreen;">@ViewBag.message</div>

<hr />

@Html.ValidationSummary(true, "", new { @class = "text-danger" })

<div class="form-group">

@Html.LabelFor(model => model.LogID, htmlAttributes: new { @class = "control-label col-md-2" })

<div class="col-md-10">

@Html.EditorFor(model => model.LogID, new { htmlAttributes = new { @class = "form-control" } })

@Html.ValidationMessageFor(model => model.LogID, "", new { @class = "text-danger" })

</div>

</div>

<div class="form-group">

@Html.LabelFor(model => model.CustEmail, htmlAttributes: new { @class = "control-label col-md-2" })

<div class="col-md-10">

@Html.EditorFor(model => model.CustEmail, new { htmlAttributes = new { @class = "form-control" } })

@Html.ValidationMessageFor(model => model.CustEmail, "", new { @class = "text-danger" })

</div>

</div>

<div class="form-group">

@Html.LabelFor(model => model.CustName, htmlAttributes: new { @class = "control-label col-md-2" })

<div class="col-md-10">

@Html.EditorFor(model => model.CustName, new { htmlAttributes = new { @class = "form-control" } })

@Html.ValidationMessageFor(model => model.CustName, "", new { @class = "text-danger" })

</div>

</div>

<div class="form-group">

@Html.LabelFor(model => model.LogStatus, htmlAttributes: new { @class = "control-label col-md-2" })

<div class="col-md-10">

@Html.EditorFor(model => model.LogStatus, new { htmlAttributes = new { @class = "form-control" } })

@Html.ValidationMessageFor(model => model.LogStatus, "", new { @class = "text-danger" })

</div>

</div>

<div class="form-group">

@Html.LabelFor(model => model.UserID, htmlAttributes: new { @class = "control-label col-md-2" })

<div class="col-md-10">

@Html.EditorFor(model => model.UserID, new { htmlAttributes = new { @class = "form-control" } })

@Html.ValidationMessageFor(model => model.UserID, "", new { @class = "text-danger" })

</div>

</div>

<div class="form-group">

@Html.LabelFor(model => model.Description, htmlAttributes: new { @class = "control-label col-md-2" })

<div class="col-md-10">

@Html.EditorFor(model => model.Description, new { htmlAttributes = new { @class = "form-control" } })

@Html.ValidationMessageFor(model => model.Description, "", new { @class = "text-danger" })

</div>

</div>

<div class="form-group">

<div class="col-md-offset-2 col-md-10">

<input type="submit" value="Save" class="btn btn-default" />

<input type="reset" value="Clear" />

</div>

</div>

</div>

}

<br>

<br>

<div>

@Html.ActionLink("Logout", "Index")

</div>

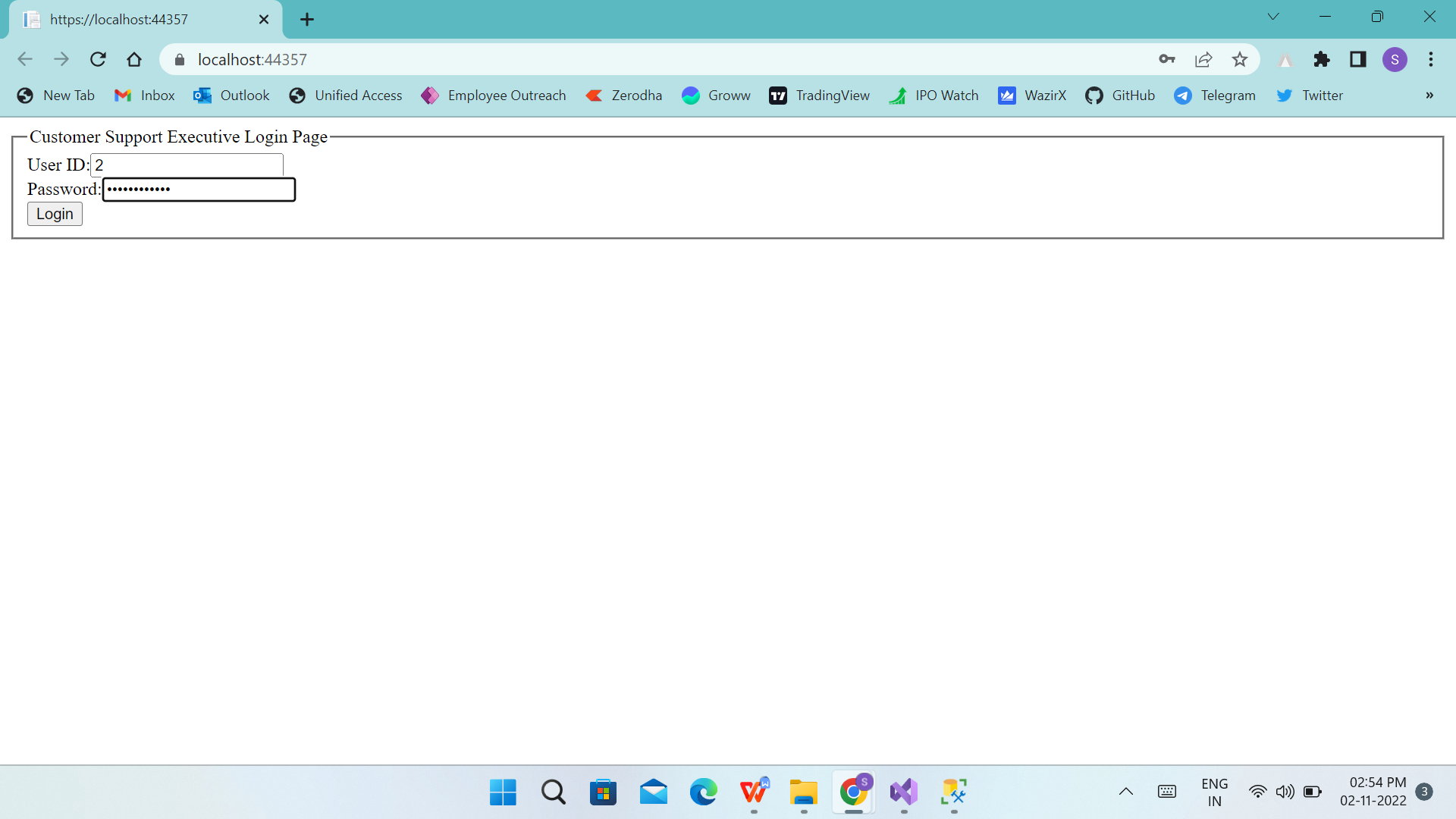
</fieldset>

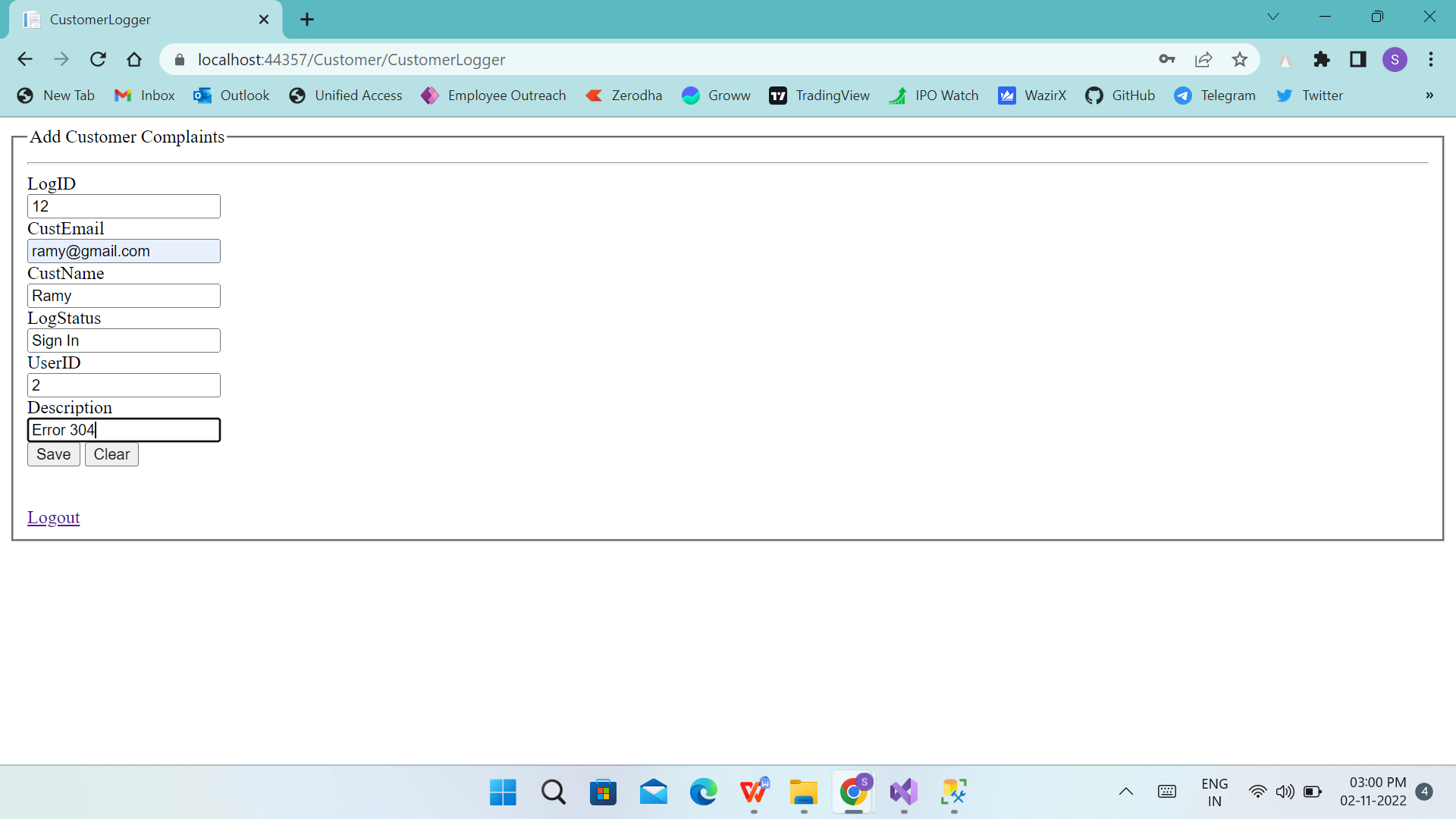
</body>

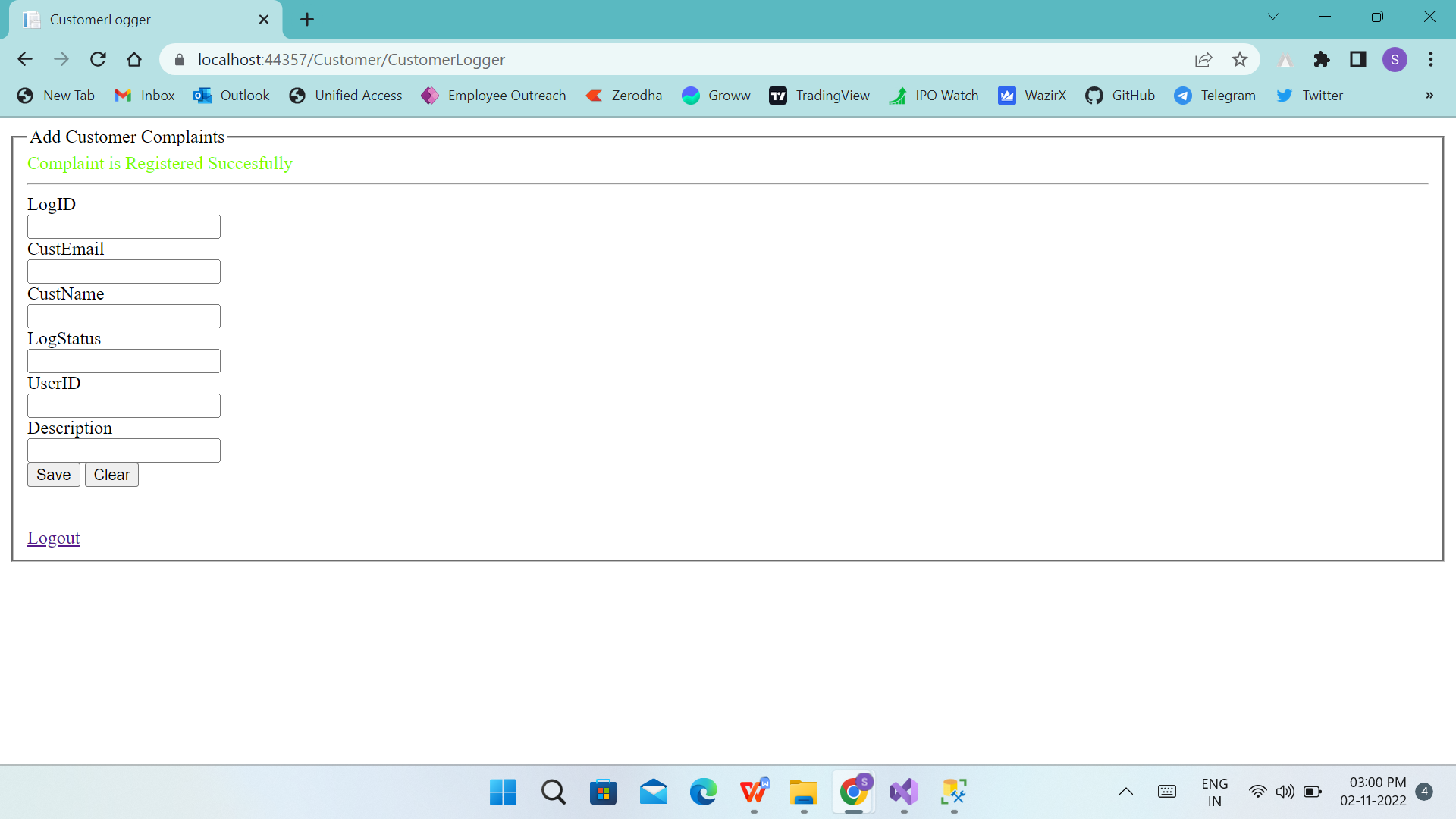
</html>

1. **Now make Customer controller as startup in route.config n Run with IIS.**

**Output:**







1. **Now add DAL.Tests project for NUnit testing n install NUnit n Nunit3Adapter to it.**

**In LoggerTest.cs**

using NUnit.Framework;

namespace CustomerDAL.Tests

{

[TestFixture]

public class LoggerTest

{

ValidateUsers v = null;

public LoggerTest()

{

v = new ValidateUsers();

}

[TestCase(1,"Hello@123456",ExpectedResult = true)]

[TestCase(2, "Hello@123123", ExpectedResult = false)]

public bool ValidateUsers(int id, string password)

{

return v.ValidateUser(id, password);

}

[TestCase(1, ExpectedResult = true)]

public bool InsertLogID(int userid)

{

CustLogInfo c = new CustLogInfo();

c.UserID = userid;

return v.Insert(c);

}

}

}

1. **Add Connection strings to App.config of Dal.Tests n Web.config also.**
2. **Run / Test in test explorer.**
3. **Now do Jenkins testing n Deploy it to Docker..**