Login page is the basic need of any Application. The user information needs to be validated in the system before doing any action in the system.We will create a login form step by step.

Create a database named ConsumerBanking.

Create a table named CBLoginInfo

```
create database ConsumerBanking

 go

USE [ConsumerBanking]
GO

/****** Object:  Table [dbo].[CBLoginInfo]    Script Date: 8/7/2016 10:06:47 PM ******/
SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO

CREATE TABLE [dbo].[CBLoginInfo](
    [CustomerId] [int] NOT NULL,
    [UserName] [nvarchar](20) NULL,
    [Password] [nvarchar](20) NULL,
    [LastLoginDate] [datetime] NULL,
    [LoginFailedCount] [int] NULL,
    [LoginIPAddress] [nvarchar](20) NULL,
    [CustomerTimeZone] [nvarchar](20) NULL,
    [LastAccessedDate] [datetime] NULL,
    [AccountLocked] [bit] NULL,
 CONSTRAINT [PK_CBLogin1] PRIMARY KEY CLUSTERED
(
    [CustomerId] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_K
) ON [PRIMARY]

GO
```

<u>Note</u>: Passwords are never stored in the plain text in any Application. It should be encrypted, so that no one can read it. For demonstration purposes, it is plain text now.

Once some records are inserted into the SQL Server database, we will write a stored procedure to fetch the user information and validate the user information.

We will create a stored procedure, named GetCBLoginInfo.

There is the logic, which we will achieve in the stored procedure, as this logic is common in any login page.

- If a user name and password is valid, it is a successful login and redirects the user to the landing page.
- If a username does not exist in the database, it shows the error 'User Does not Exist'.
- If the user is a valid user and wrong password is given by the user, it will give the message, number of failed attempts.
- If failed attempt is more than or equal to 5 times, it will lock the user out.

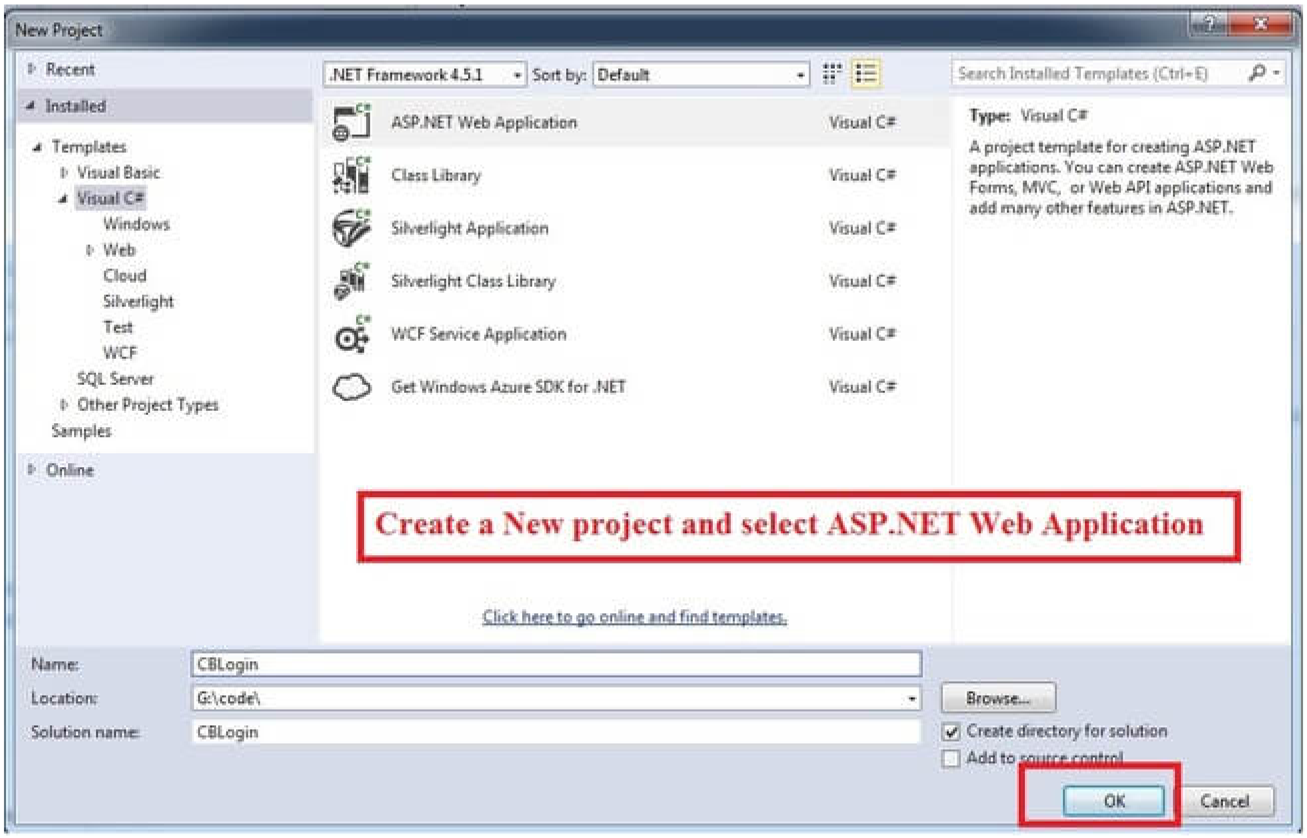
```
USE [ConsumerBanking]
GO
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
```

```
GO

CREATE  PROCEDURE [dbo].[GetCBLoginInfo]
    @UserName VARCHAR(20),
    @Password varchar(20)
AS
SET NOCOUNT ON
Declare @Failedcount AS INT
SET @Failedcount = (SELECT LoginFailedCount from CBLoginInfo WHERE

IF EXISTS(SELECT * FROM CBLoginInfo WHERE UserName = @UserName)

 BEGIN

 IF EXISTS(SELECT * FROM CBLoginInfo WHERE UserName = @UserName AN
    SELECT 'Success' AS UserExists
ELSE

Update CBLoginInfo set  LoginFailedCount = @Failedcount+1  WHERE U

Update CBLoginInfo set LastLoginDate=GETDATE()  WHERE UserName = @
 BEGIN
IF @Failedcount >=5


SELECT 'Maximum Attempt Reached (5 times) .Your Account is locked
ELSE

select CONVERT(varchar(10), (SELECT LoginFailedCount from CBLoginI
END
END
 ELSE

 BEGIN

SELECT 'User Does not Exists' AS UserExists
 END
```

We have created the stored procedure. Our database part is ready now.

Now, we will create ASP.NET MVC Web Application to create a login page .We will call the stored procedure, using Entity framework to validate the user information from the database.

Create a new project and select ASP.NET Web Application. Click OK.



New Project

Recent / Installed / Templates / Visual Basic / Visual C# / Windows / Web / Cloud / Silverlight / Test / WCF / SQL Server / Other Project Types / Samples / Online

.NET Framework 4.5.1 · Sort by: Default · Search Installed Templates (Ctrl+E)

ASP.NET Web Application	Visual C#
Class Library	Visual C#
Silverlight Application	Visual C#
Silverlight Class Library	Visual C#
WCF Service Application	Visual C#
Get Windows Azure SDK for .NET	Visual C#

Type: Visual C#

A project template for creating ASP.NET applications. You can create ASP.NET Web Forms, MVC, or Web API applications and add many other features in ASP.NET.

Create a New project and select ASP.NET Web Application

Click here to go online and find templates.

Name: CBLogin

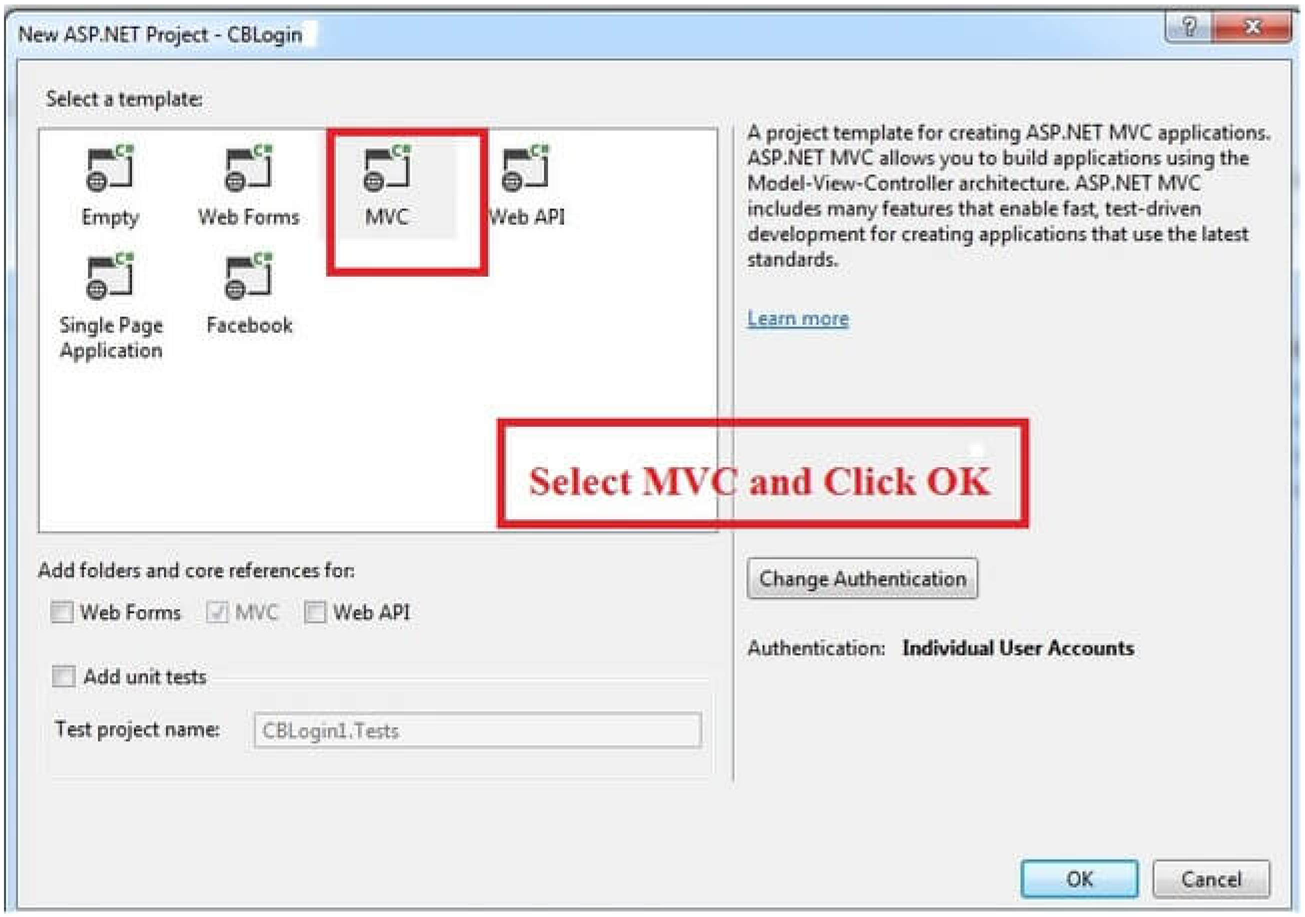
Location: G:\code\

Solution name: CBLogin

[✓] Create directory for solution

[ ] Add to source control

Select MVC and click OK.



New ASP.NET Project - CBLogin

Select a template:

Empty · Web Forms · MVC · Web API · Single Page Application · Facebook

A project template for creating ASP.NET MVC applications. ASP.NET MVC allows you to build applications using the Model-View-Controller architecture. ASP.NET MVC includes many features that enable fast, test-driven development for creating applications that use the latest standards.

Learn more

Select MVC and Click OK

Add folders and core references for:

[ ] Web Forms [✓] MVC [ ] Web API

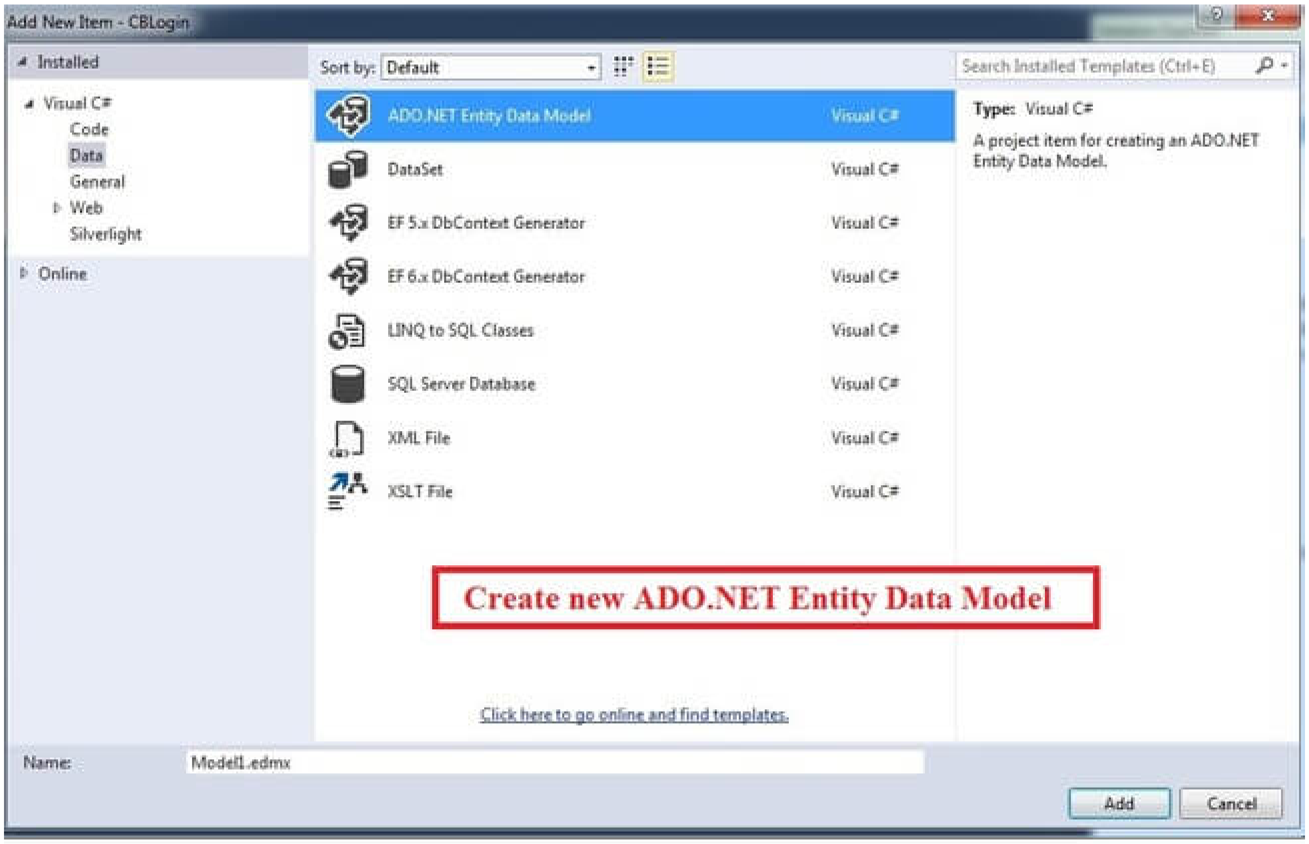
[ ] Add unit tests

Test project name: CBLogin1.Tests

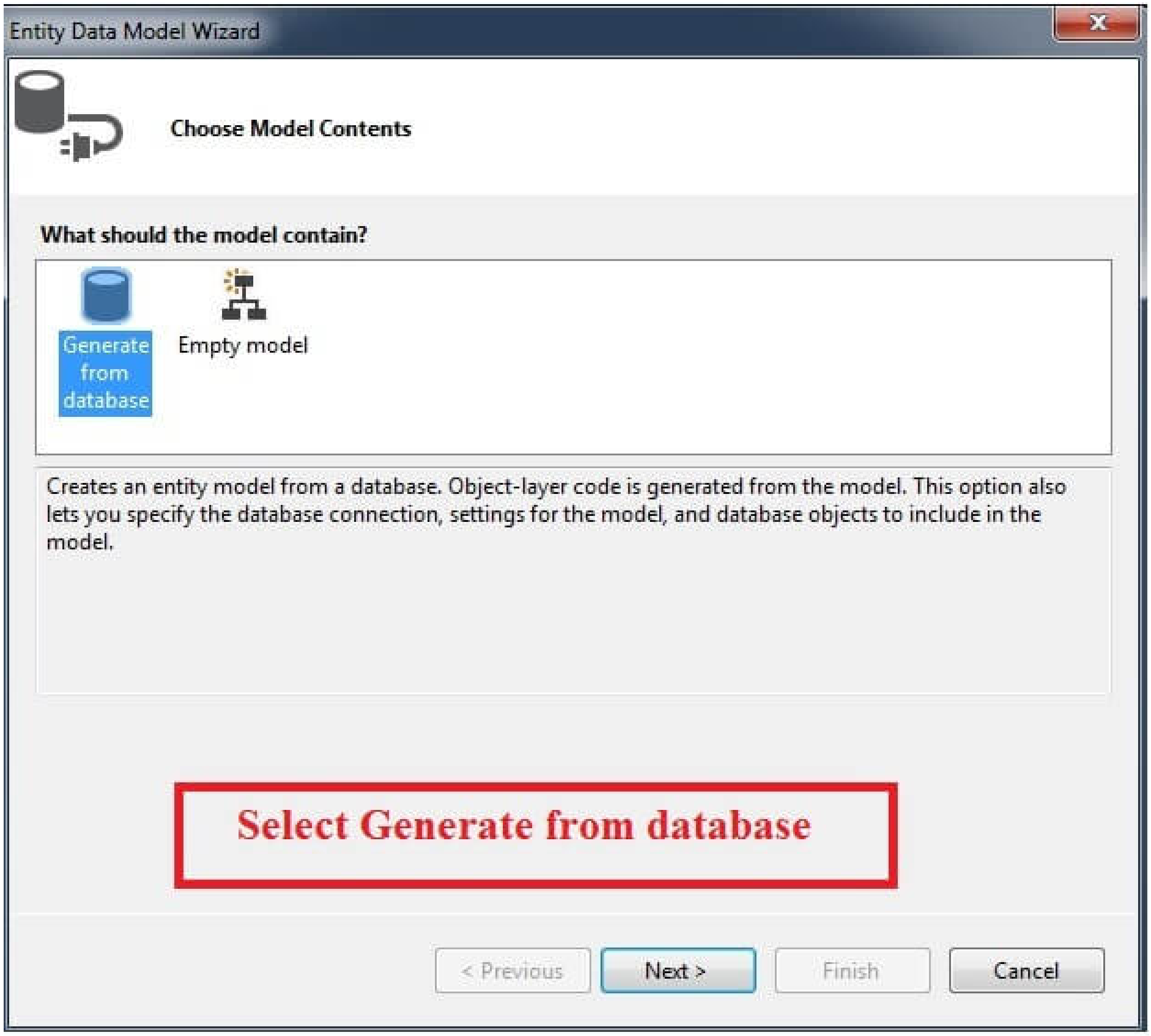
Authentication: **Individual User Accounts**

The MVC project is created now.

We will use Entity Framework as a data fetching layer. We will add an EDMX file to fetch the data from the database. We will call the stored procedure, which we created earlier.

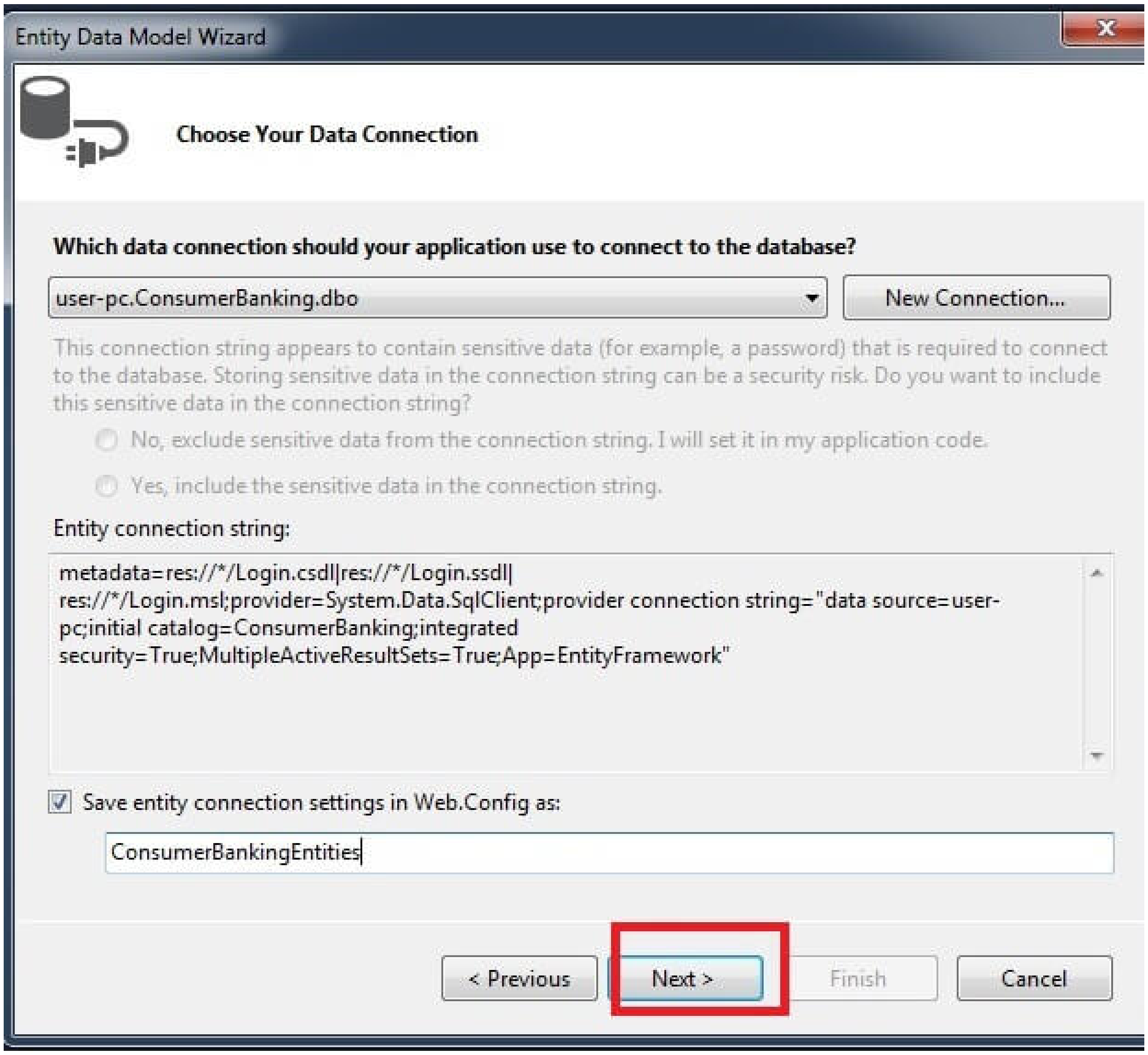

Add New Item - CBLogin
Installed
Visual C#
Code
Data
General
Web
Silverlight
Online
Sort by: Default
Search Installed Templates (Ctrl+E)
ADO.NET Entity Data Model Visual C#
DataSet Visual C#
EF 5.x DbContext Generator Visual C#
EF 6.x DbContext Generator Visual C#
LINQ to SQL Classes Visual C#
SQL Server Database Visual C#
XML File Visual C#
XSLT File Visual C#
Type: Visual C#
A project item for creating an ADO.NET Entity Data Model.
Create new ADO.NET Entity Data Model
Click here to go online and find templates.
Name: Model1.edmx
Add
Cancel

Select the option Generate from the database.



Creates an entity model from a database. Object-layer code is generated from the model. This option also lets you specify the database connection, settings for the model, and database objects to include in the model.

Select your database Server and the database tables in the next step.


Entity Data Model Wizard

Choose Your Data Connection

Which data connection should your application use to connect to the database?

user-pc.ConsumerBanking.dbo

New Connection...

This connection string appears to contain sensitive data (for example, a password) that is required to connect to the database. Storing sensitive data in the connection string can be a security risk. Do you want to include this sensitive data in the connection string?

No, exclude sensitive data from the connection string. I will set it in my application code.

Yes, include the sensitive data in the connection string.

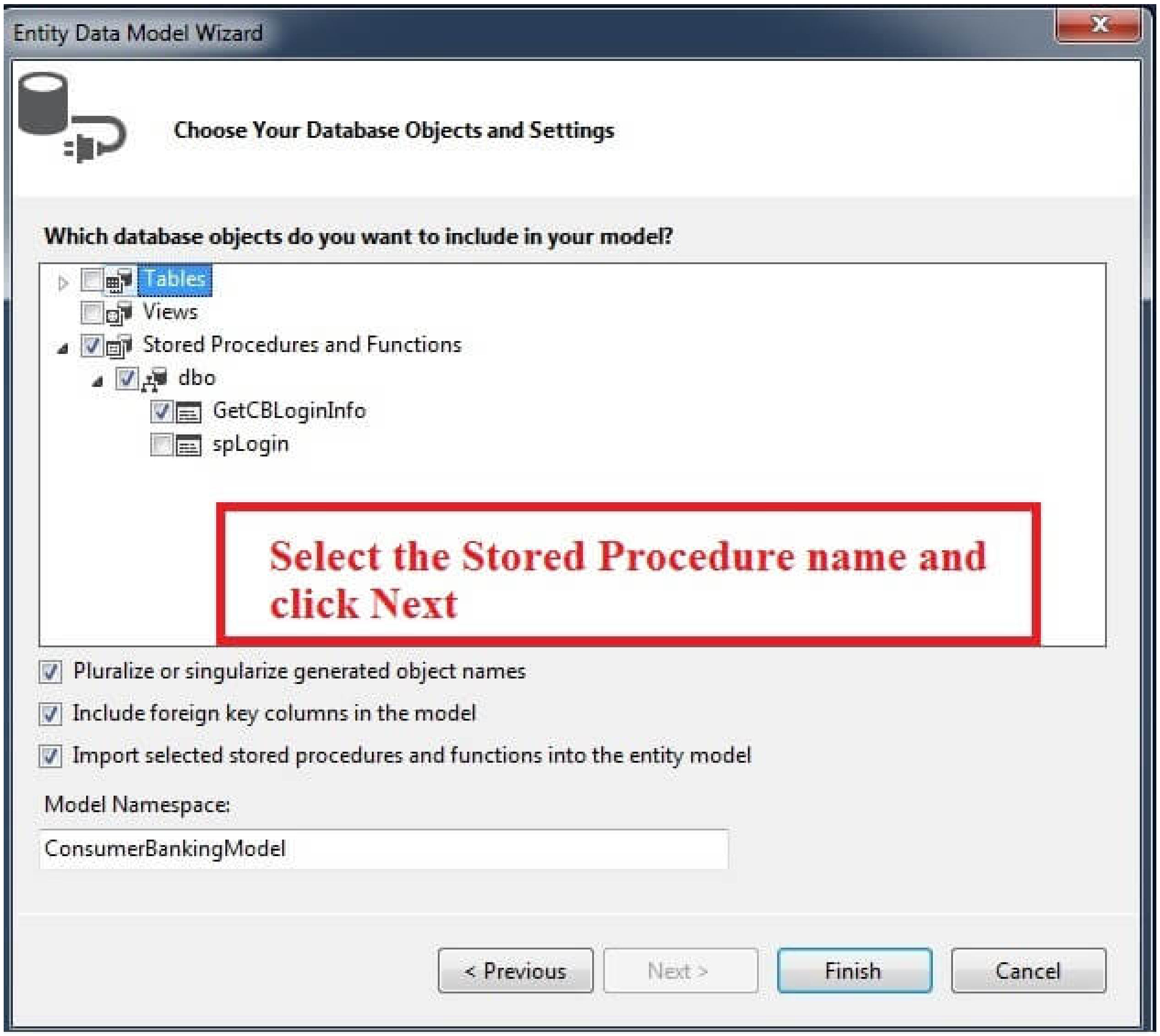
Entity connection string:

```
metadata=res://*/Login.csdl|res://*/Login.ssdl|
res://*/Login.msl;provider=System.Data.SqlClient;provider connection string="data source=user-
pc;initial catalog=ConsumerBanking;integrated
security=True;MultipleActiveResultSets=True;App=EntityFramework"
```

[✓] Save entity connection settings in Web.Config as:

ConsumerBankingEntities

Select the stored procedure in the next step.



Which database objects do you want to include in your model?

- [ ] Tables
- [ ] Views
- [✓] Stored Procedures and Functions
  - [✓] dbo
    - [✓] GetCBLoginInfo
    - [ ] spLogin

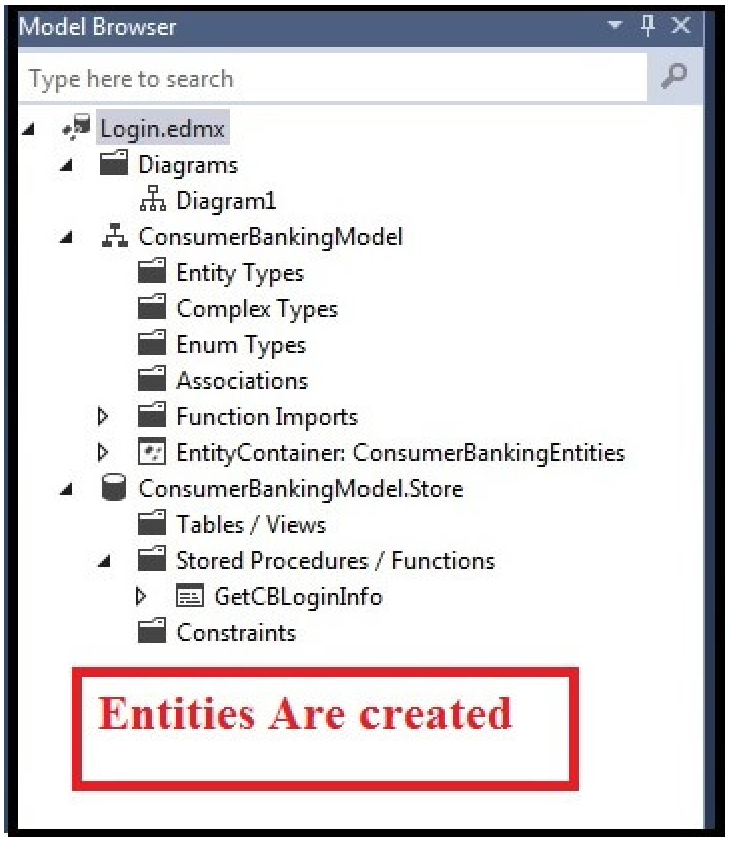
Select the Stored Procedure name and click Next

- [✓] Pluralize or singularize generated object names
- [✓] Include foreign key columns in the model
- [✓] Import selected stored procedures and functions into the entity model

Model Namespace:

ConsumerBankingModel

Click Finish. Now, ADO.NET Entity Data Model is created for us.


Model Browser

Type here to search

Login.edmx
Diagrams
Diagram1
ConsumerBankingModel
Entity Types
Complex Types
Enum Types
Associations
Function Imports
EntityContainer: ConsumerBankingEntities
ConsumerBankingModel.Store
Tables / Views
Stored Procedures / Functions
GetCBLoginInfo
Constraints

Entities Are created

We will create a new model class, named CBUserModel, which has two properties, named UserName and Password. This model class will be used to communicate between the view and controller. We have some basic validation to validate the user Name and the Password fields will not to be blank, using DataAnnotations from the System.ComponentModel.

```
using System.ComponentModel.DataAnnotations;

namespace CBLogin.Models
{
    public class CBUserModel
    {
        [Required(ErrorMessage = "UserName is required")]
        public string UserName { get; set; }
        [Required(ErrorMessage = "Password is required")]
        [DataType(DataType.Password)]
        public string Password { get; set; }
    }
}
```

We will create a controller, named CBLoginController, which has the actions, given below:

The Index view will return us the Index view.

```
public ActionResult Index()
      {
          return View();
      }
```

This Index action with [Httppost] verb will be called, when the user posts the data after entering UserName and Password field. In this action, the Username and Password will be validated against the database.

```
[HttpPost]
```

```
public ActionResult Index(CBUserModel model)
{
    ConsumerBankingEntities cbe = new ConsumerBankingEntities();
    var s = cbe.GetCBLoginInfo(model.UserName, model.Password);

    var item = s.FirstOrDefault();
    if (item == "Success")
    {

        return View("UserLandingView");
    }
    else if(item=="User Does not Exists")

    {
        ViewBag.NotValidUser = item;

    }
    else
    {
        ViewBag.Failedcount = item;
    }

  return View("Index");
}
```

The action UserLandingView will be called, when the user posts the data after entering UserName and Password field. There is a successful login.

```
public ActionResult UserLandingView()
    {
        return View();
    }
```

Index View

In the Index view, we have two input textbox fields, named UserName, Password and a Login button.

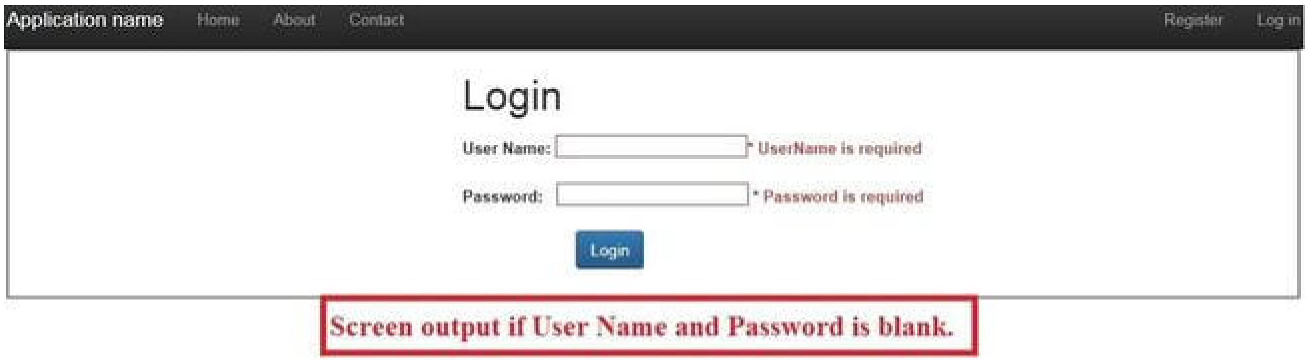
```
@model CBLogin.Models.CBUserModel

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-
scale=1">

    <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/b
    <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/b
theme.min.css">
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/1.12
</script>
    <script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/j
</script>
    <style type="text/css">
        .bs-example {

            height:220px;

        }

        .centerlook {
            padding-left: 400px;
            font-weight: bold;
            width: 1000px;
        }
        .error {
            padding-left: 400px;
            font-weight: bold;
            width: 1000px;
            color: red;
        }

        .loginbtn {
            padding-left: 500px;
        }
    </style>

</head>
<body>

    @using (Html.BeginForm())
    {
        <div class="bs-example" style="border:2px solid gray;">

            <div class="form-group centerlook">
                <h1> Login </h1>
            </div>
            <div class="form-group centerlook">
                <label>User Name: </label>
               @Html.EditorFor(model => model.UserName)*
                @Html.ValidationMessageFor(model => model.UserName
            </div>
            <div class="form-group centerlook">
                <label>Password:</label>
                   @Html.EditorFor(model => model.Password) *
                @Html.ValidationMessageFor(model => model.Passworc

            </div>
            <div class="form-group error">
                @if (@ViewBag.Failedcount != null)
                {

                    <label> Failed Attempt count is: @ViewBag.Fail
                }

                @if (@ViewBag.NotValidUser != null)
                {

                    <label> @ViewBag.NotValidUser</label>
                }
            </div>
            <div class="loginbtn">

                <input type="submit" value="Login" class="btn btn-primary" />

            </div>
        </div>
    }
</body>
</html>
```

When a user loads the Index action, the Index view will be loaded. When the user enters UserName, Password and clicks the Login button, the Index action with HttpPost attribute is called.

The Entity framework code validates the username and password, given below. Based on the status returned from the stored procedure, the user will be shown an error message or redirected to the landing page.

When we run the page, we get the output of the page, as we have stated in the starting of the topic.

Condition 1: If the User Name and Password is blank, it will show the Validation Error Message, as shown in the screen, given below. We can add regular expression and other validation with the help of Component Model .



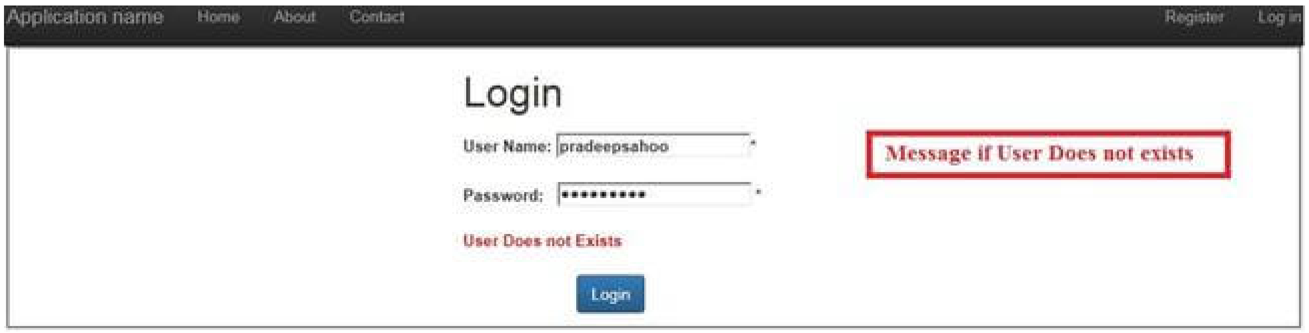
Login

User Name: * UserName is required

Password: * Password is required

Screen output if User Name and Password is blank.

Condition 2: If a User Name and Password is given by the user and UserName does not exist in the database /system, it shows message to the user “User Does not Exists”, as shown in the screen, given below:



Login

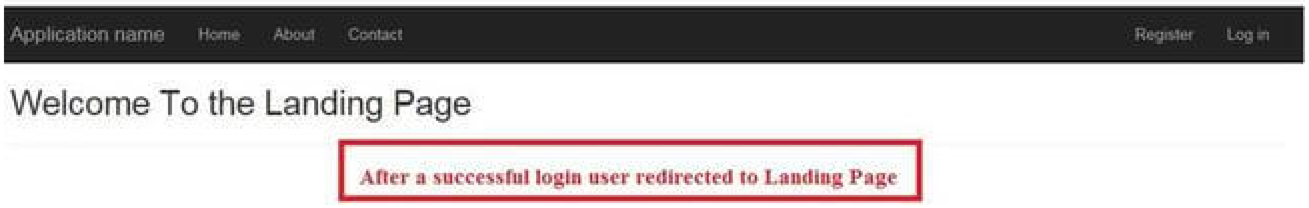
User Name: pradeepsahoo *

Password: •••••••••• *

User Does not Exists

Message if User Does not exists

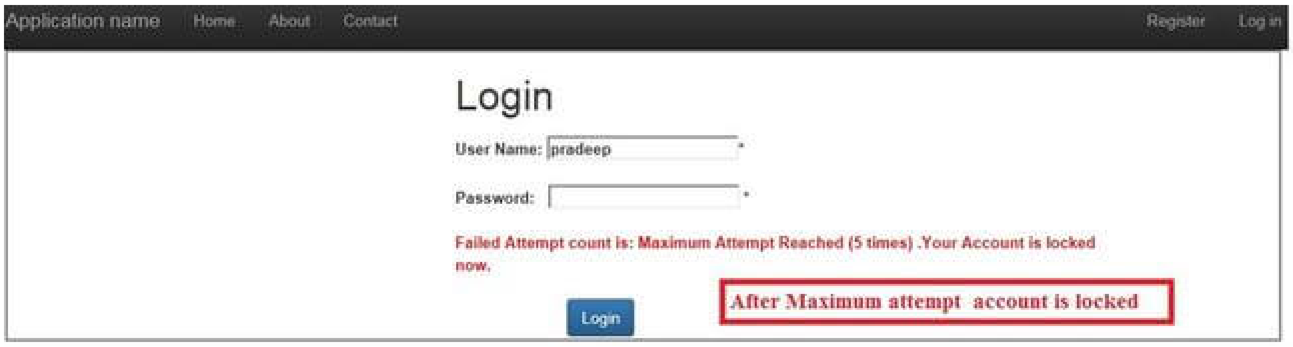
Condition 3: If the User Name and Password given by the user is a valid user and username and password is correct, the user will be navigated to the landing page view.



Welcome To the Landing Page

After a successful login user redirected to Landing Page

Condition 4: If the User Name and Password is given wrong for five times or more than five times, then the user account will be locked, as shown in the screen, given below:



Login

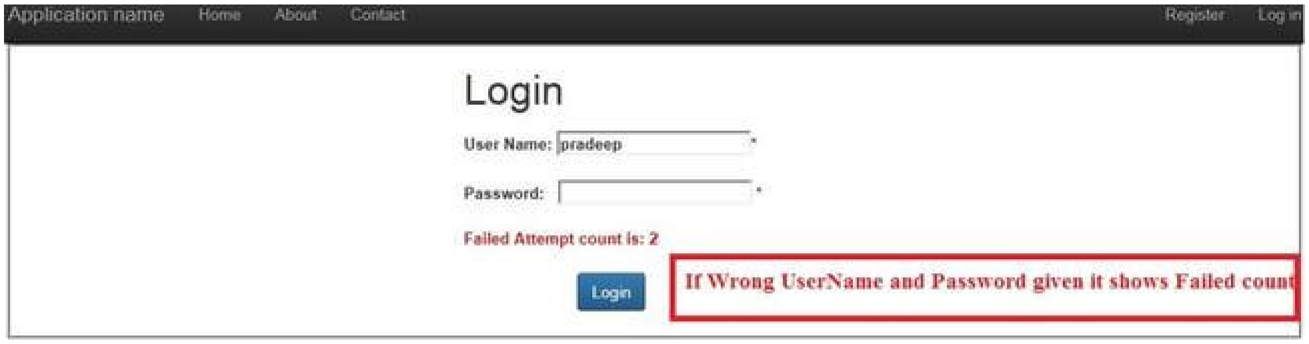
User Name: pradeep *

Password: *

Failed Attempt count is: Maximum Attempt Reached (5 times) .Your Account is locked now.

After Maximum attempt account is locked

Condition 5: If the User Name and Password is given wrong, the page will display the number of failed attempts done by the user .



Login

User Name: pradeep *

Password: *

Failed Attempt count is: 2

If Wrong UserName and Password given it shows Failed count