https://www.halvorsen.blog

ASP.NET Core

User Identity and Login

Hans-Petter Halvorsen

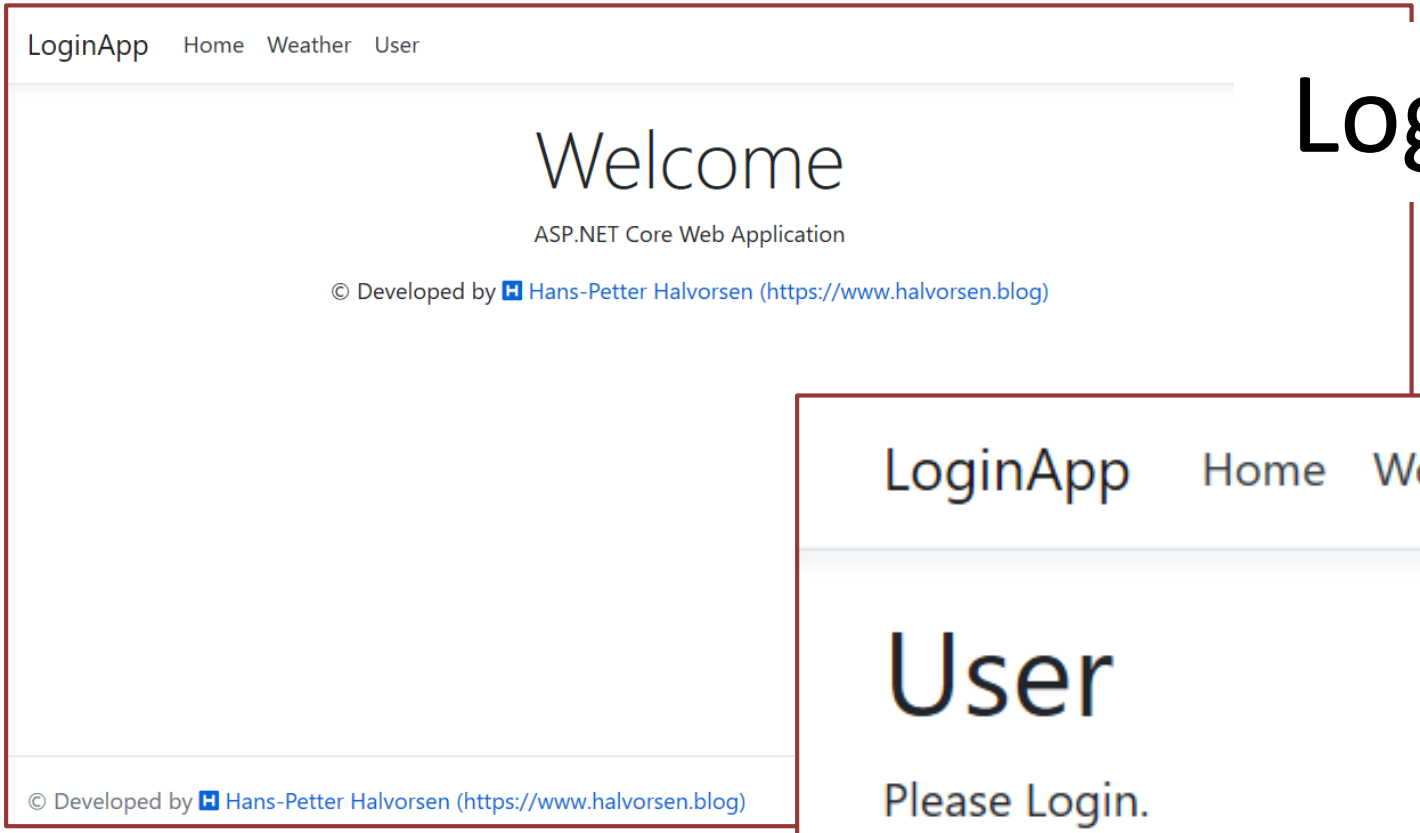
ASP.NET Core

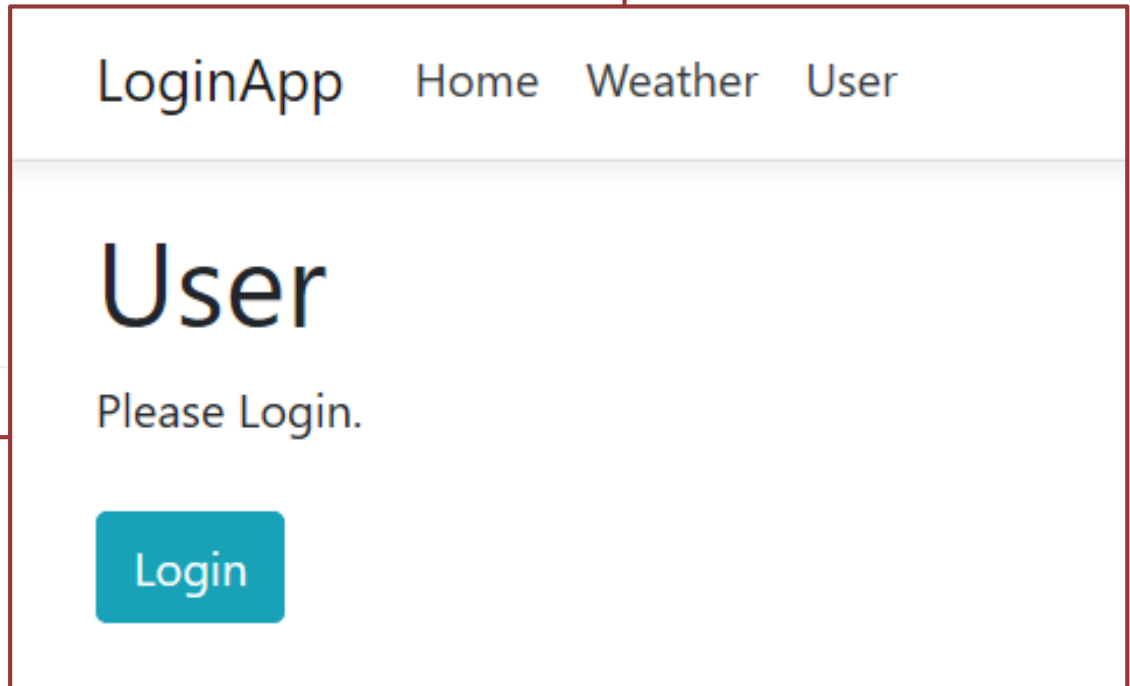
If you have never used ASP.NET Core, I suggest the following Videos:

- ASP.NET Core - Hello World
  https://youtu.be/lcQsWYgQXK4
- ASP.NET Core – Introduction
  https://youtu.be/zkOtiBcwo8s

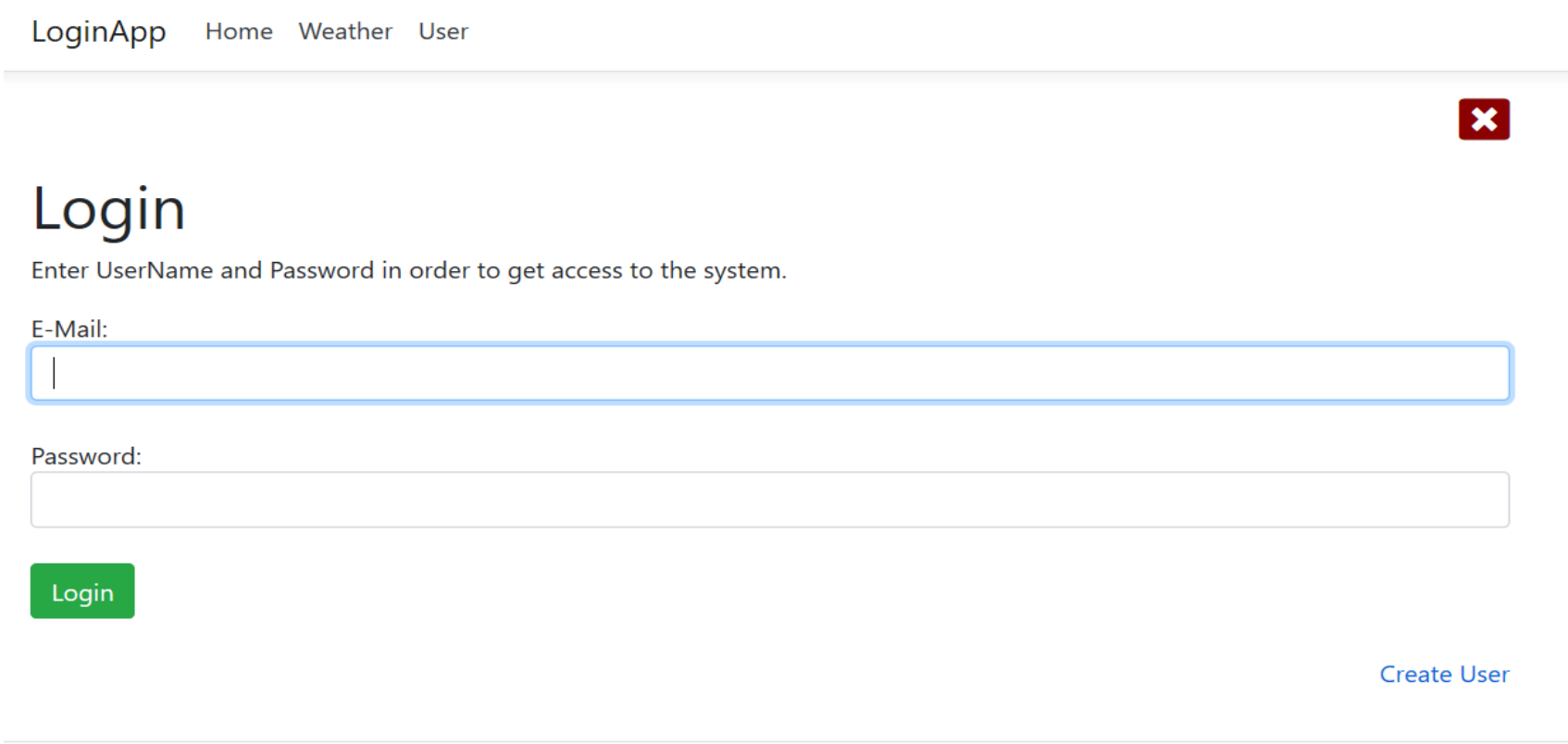
ASP.NET Core Resources:
https://halvorsen.blog/documents/programming/web/aspnet

Login App

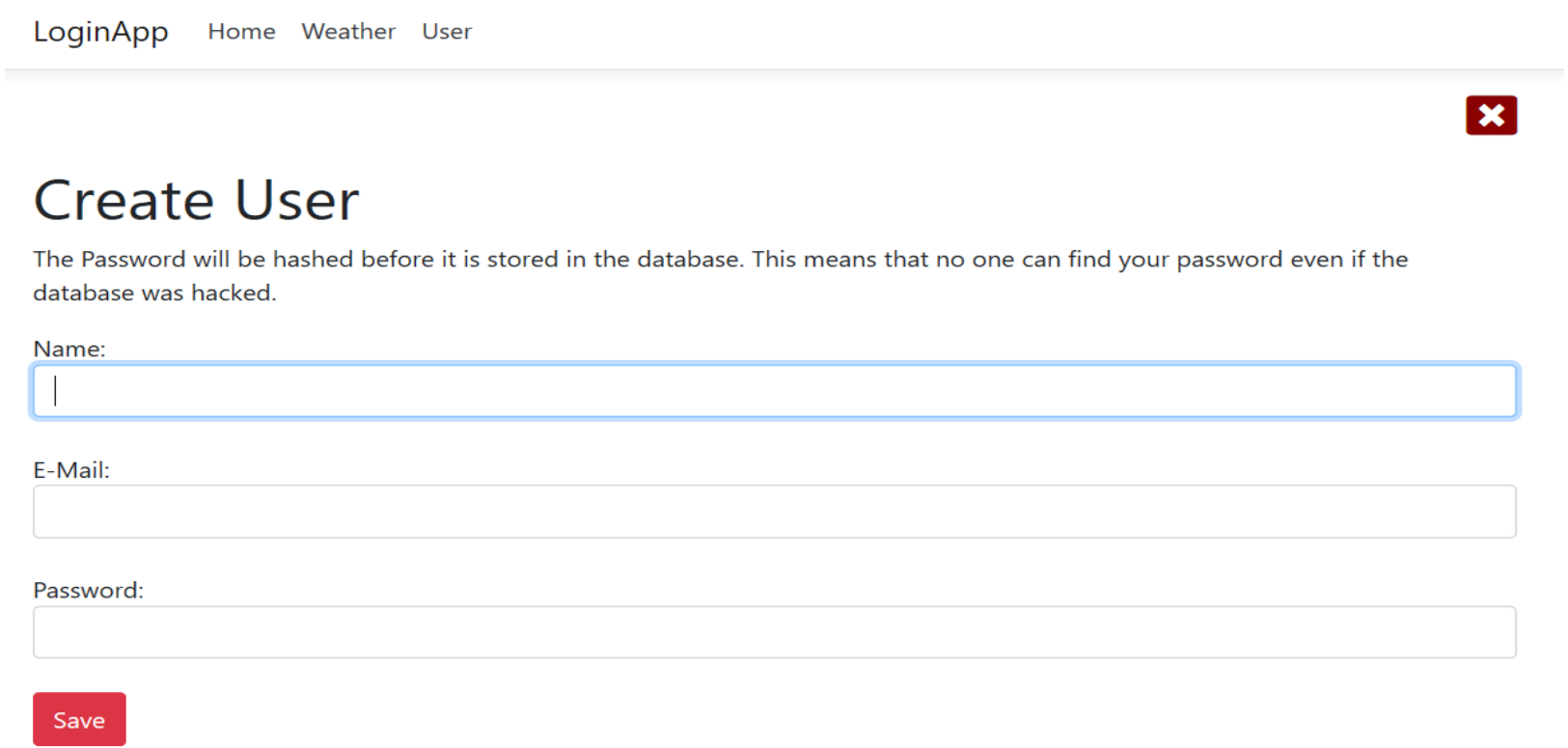

LoginApp
Home
Weather
User
Welcome
ASP.NET Core Web Application
© Developed by Hans-Petter Halvorsen (https://www.halvorsen.blog)
© Developed by Hans-Petter Halvorsen (https://www.halvorsen.blog)


LoginApp
Home
Weather
User
User
Please Login.
Login

Login

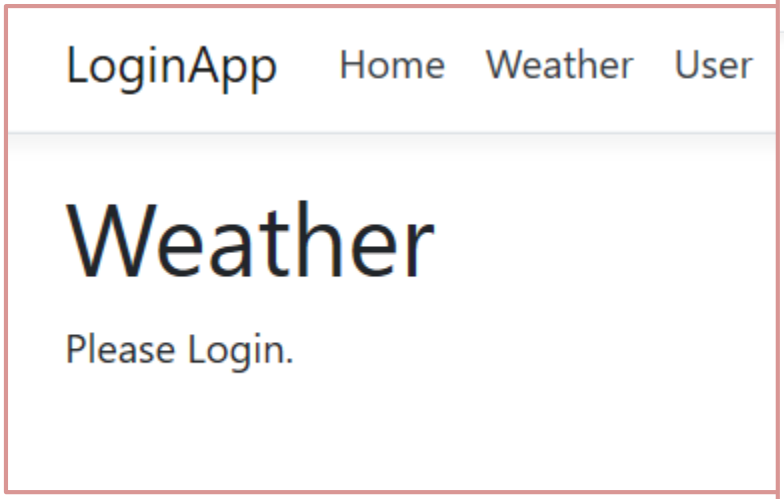

LoginApp
Home
Weather
User
Login
Enter UserName and Password in order to get access to the system.
E-Mail:
Password:
Login
Create User
© Developed by Hans-Petter Halvorsen (https://www.halvorsen.blog)

Create User


LoginApp
Home
Weather
User
Create User
The Password will be hashed before it is stored in the database. This means that no one can find your password even if the database was hacked.
Name:
E-Mail:
Password:
Save
© Developed by Hans-Petter Halvorsen (https://www.halvorsen.blog)

Weather Data

Available only when user is logged in



LoginApp Home Weather User

Weather

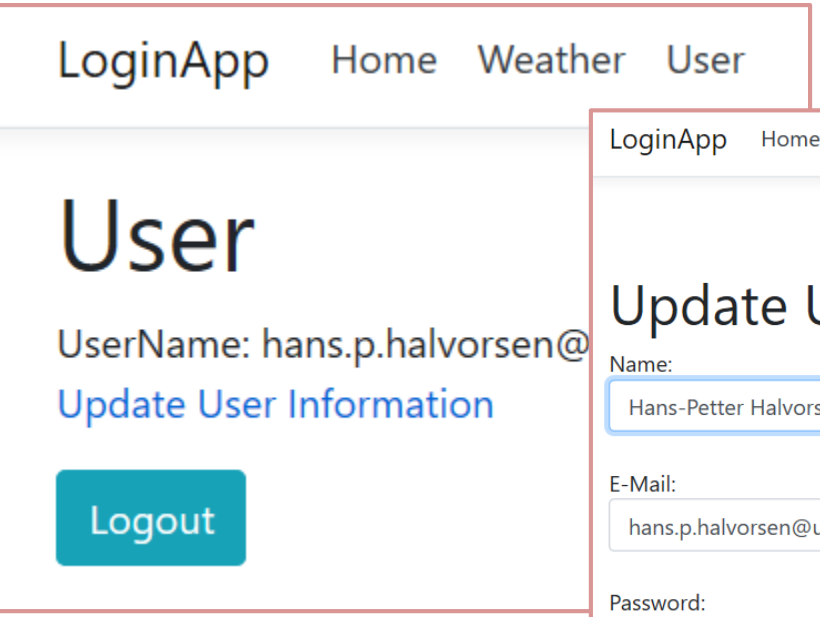
UserName: hans.p.halvorsen@usn.no

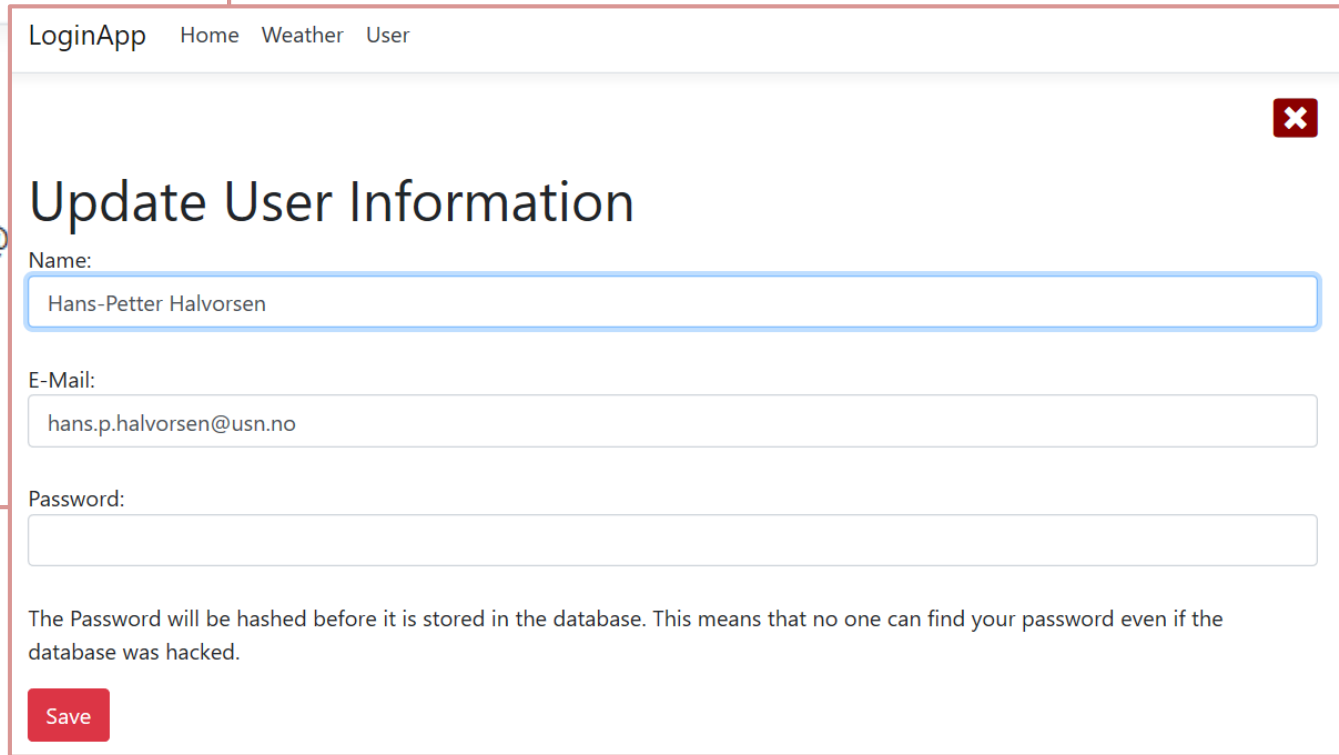
Below you see Weather Data

Parameter	Value	Unit	TimeStamp
Temperature	22	°C	2020-03-04 11:01:53
Temperature	21	°C	2020-03-04 11:01:53
Temperature	23	°C	2020-03-04 11:01:53
Temperature	19	°C	2020-03-04 11:01:53
Temperature	18	°C	2020-03-04 11:01:53
Temperature	20	°C	2020-03-04 11:01:53
Temperature	24	°C	2020-03-04 11:01:53

Update User Information

Available only when user is logged in





Password Security

- Encryption and Decrypting
- Hashing
- Salting
- 2 Factor Authentication
- Etc.

Encryption and Decryption

- Encryption is the practice of scrambling information in a way that only someone with a corresponding key can unscramble and read it.
- Encryption is a two-way function.
- When you encrypt something, you’re doing so with the intention of decrypting it later.
- To encrypt data you use an algorithm. Many different encryption algorithms do exist

Encryption and Decryption

Plain Text

Encrypted Text

Plain Text

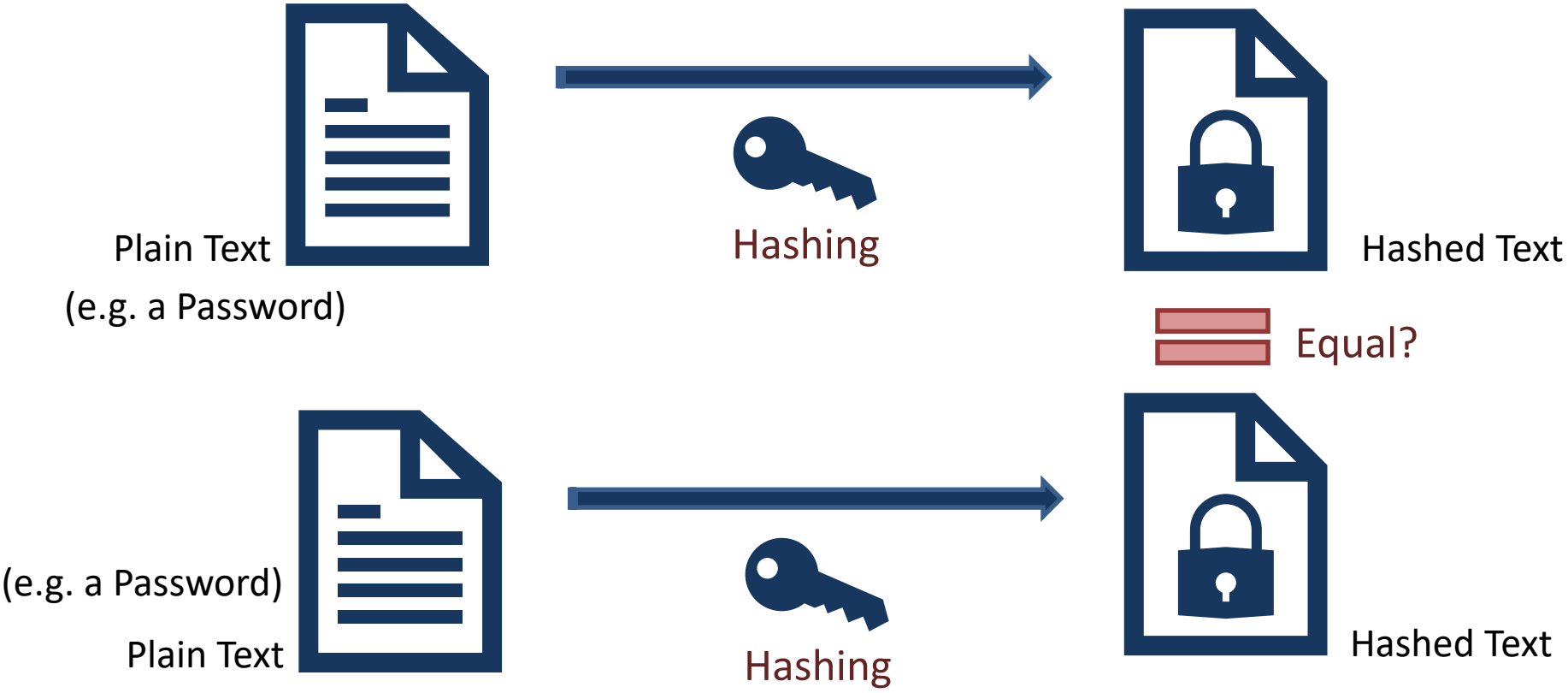
When should encryption be used?

- Encryption is a two-way function.
- You encrypt information with the intention of decrypting it later.
- Examples when to use encryption:
  - Protecting Files and Information o your Computer
  - Protecting your Cloud data
  - Transmitting Data between 2 Computers
  - Etc.
- The key is that Encryption is reversible. Hashing is not.

Encryption and Hashing

- Hashing is the practice of using an algorithm to map data of any size to a fixed length.
- Encryption is a two-way function
- Hashing is a one-way function.
- While it's technically possible to reverse-hash something, the computing power required makes it unfeasible. Hashing is one-way.
- Encryption is meant to protect data in transit, hashing is meant to verify that a file or piece of data hasn't been altered—that it is authentic. In other words, it serves as a check-sum.
- Every hash value is unique

Hashing


Plain Text
(e.g. a Password)
Hashing
Hashed Text
Equal?
(e.g. a Password)
Plain Text
Hashing
Hashed Text

Hacking Hashing?

Password Table for System X

UserName	HashedPassword
Mike	4420d1918bbcf7
Bob	73fb51a0c9be7d
Peter	4420d1918bbcf7

If a Hacker gets access to this Database, he can see that Mike and Peter have the same password.
But he does not know the actual password

Password	HashedPassword
tesla	4420d1918bbcf7
friendship	73fb51a0c9be7d
bicycle	7420e1618abcf6

Rainbow table

If the Hacker has access to so-called "Rainbow table" (which is essentially a pre-computed database of hashes), he may also be able to find the Password (as seen here)

If you have a complicated password, it is less likely that your password is in such a Rainbow table

Salting

- Salting is a technique typically used for Password Hashing.
- It is a unique value that can be added to the end of the password to create a different hash value.
- The additional value is referred to as a “salt”.
- This is done to make it even more secure.
- Typically, the Hashing Algorithm uses a Random salt.
  - This prevents an attacker from seeing whether users have the same password.

Salting

```
password = "Password123"
salt = "Tesla"

passwordHashed = HashPassword(password, salt);
```

Typically, Salting is built into the Hashing Algorithm and it is changed every time

```
password = "Password123"

ph1 = HashPassword(password);
ph2 = HashPassword(password);
```

ph1  ph2

This means if 2 different Users use the same Password, the Hashed Password will be different!

Hacking Hashing with Salt?

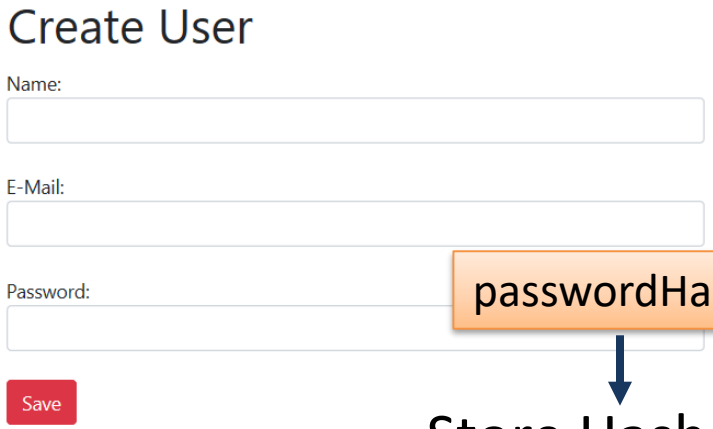
Assume Mike and Peter use the same Password

UserName	HashedPasswordwithSalt
Mike	4420d1918bbcf7
Bob	73fb51a0c9be7d
Peter	4520d1818cbcf7

If a Hacker gets access to this Database, he cannot see that Mike and Peter have the same password.

Because a random Salt has made these 2 Hashed Passwords different!

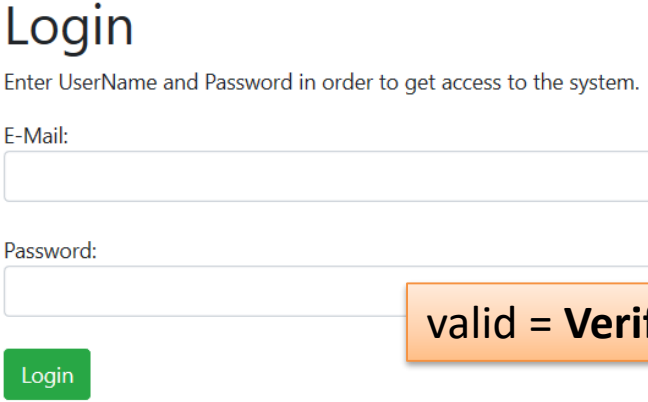
Create User and Login



Information given by User

passwordHashed = **HashPassword**(userName, password);

Store Hashed Password in the Database



Compare Hashed Password stored in the Database with Password given by User in Login Page

valid = **VerifyHashedPassword**(userName, passwordDB, password);

Microsoft.AspNetCore.Identity

- This Namespace contains different Classes and Methods for Identity handling
- We will use the **PasswordHasher<TUser>** Class

PasswordHasher<TUser> Class

Namespace: Microsoft.AspNetCore.Identity

Methods:

- **HashPassword**(TUser, String)
  - Returns a hashed representation of the supplied password for the specified user.
- **VerifyHashedPassword**(TUser, String, String)
  - Returns a PasswordVerificationResult indicating the result of a password hash comparison.

Microsoft.AspNetCore.Identity

Example:

```
using Microsoft.AspNetCore.Identity;
…
string username;  //UserName given by user when creating a User
string passwordHashed;

PasswordHasher<string> pw = new PasswordHasher<string>();

passwordHashed = pw.HashPassword(userName, password);
```

Session Data

- We need to store information whether the User is logged in or not
- We can use Session variables in order to share that information between multiple web pages

Session State in ASP.NET Core

Session management in ASP.NET Core is not enabled by default.

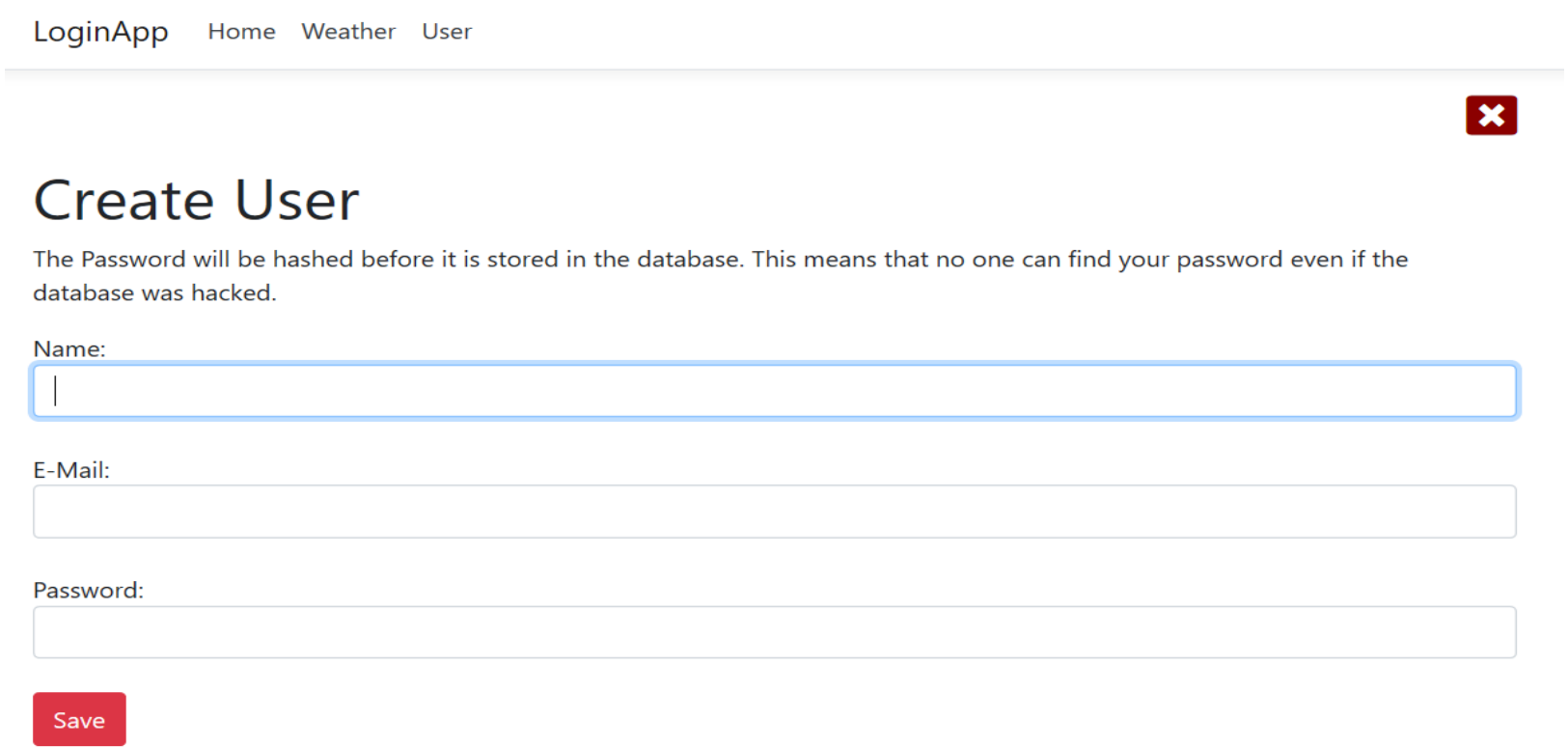
- You need to install the **Microsoft.AspNetCore.Session** NuGet Package in order to use Session state.
- You need to **enable Session State** in the **Startup.cs** file
- You need to include the Namespace using **Microsoft.AspNetCore.Http**;

https://www.halvorsen.blog

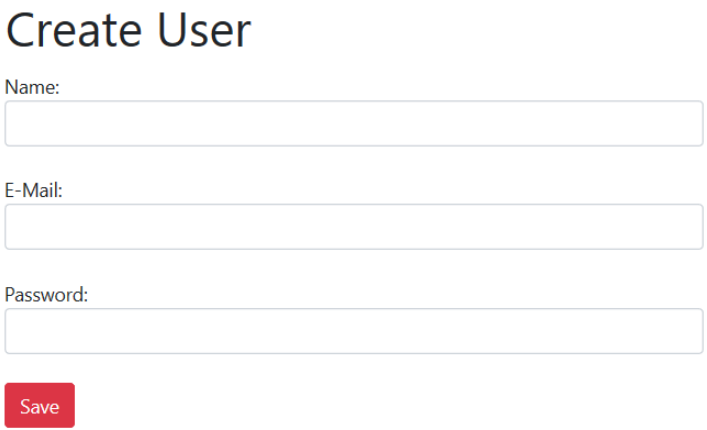
Create User

Hans-Petter Halvorsen

Create User


LoginApp
Home
Weather
User
Create User
The Password will be hashed before it is stored in the database. This means that no one can find your password even if the database was hacked.
Name:
E-Mail:
Password:
Save

Create User



Information given by User

passwordHashed = **HashPassword**(userName, password);

Store Hashed Password in the Database

Web Page (**CreateUser.cshtml**)

```
<div>
    <h1>Create User</h1>

    <form method="post">

        Name:
        <br />
        <input name="FullName" type="text" class="form-control input-lg" required autofocus/>
        <br />

        E-Mail:
        <br />
        <input name="EMail" type="email" class="form-control input-lg" required />
        <br />

        Password:
        <br />
        <input name="Password" type="password" class="form-control input-lg" required />
        <br />

        <input id="SaveButton" type="submit" value="Save" class="btn btn-danger">

    </form>

</div>
```

Code (**CreateUser.cshtml.cs**)

```
...
using Microsoft.AspNetCore.Identity;
...
    void CreateUser()
    {
        Person person = new Person();
        string userName;
        string password;

        connectionString = _configuration.GetConnectionString("ConnectionString");

        person.FullName = Request.Form["FullName"];
        userName = Request.Form["EMail"];
        password = Request.Form["Password"];

        person.EMail = userName;
        person.Password = PasswordHash(userName, password);

        person.CreateUser(connectionString, person);  //Store User Information in Database, including Hashed Password


        Response.Redirect("./User");
    }


    string PasswordHash(string userName, string password)
    {

        PasswordHasher<string> pw = new PasswordHasher<string>();

        string passwordHashed = pw.HashPassword(userName, password);

        return passwordHashed;

    }
```

Person.cs Class

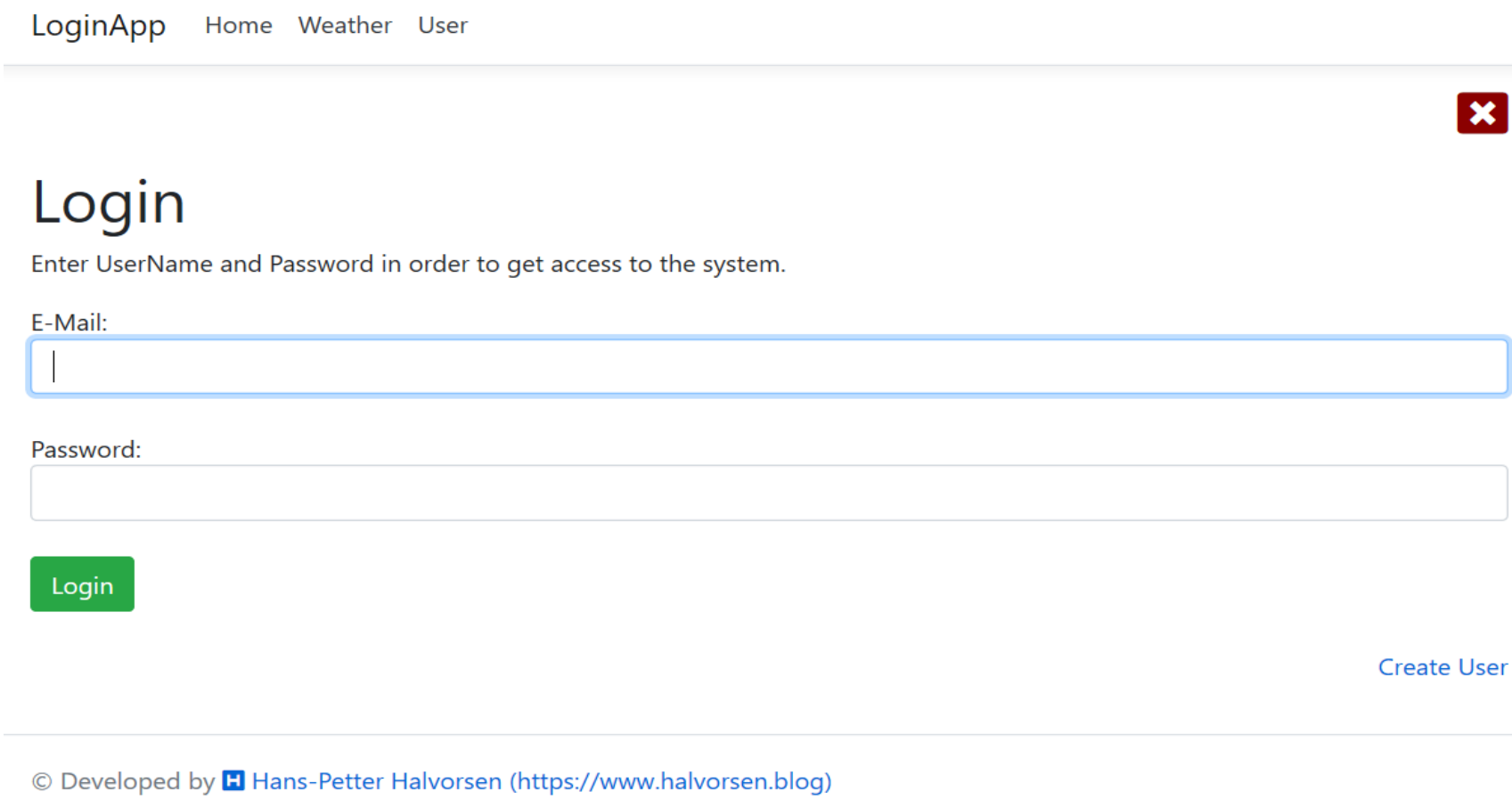
```
public class Person
{
    public string FullName { get; set; }
    public string EMail { get; set; }
    public string Password { get; set; }

    public void CreateUser(string connectionString, Person person)
    {
        try
        {
            using (SqlConnection con = new SqlConnection(connectionString))
            {
                SqlCommand cmd = new SqlCommand("CreateUser", con);
                cmd.CommandType = CommandType.StoredProcedure;

                cmd.Parameters.Add(new SqlParameter("@FullName", person.FullName));
                cmd.Parameters.Add(new SqlParameter("@EMail", person.EMail));
                cmd.Parameters.Add(new SqlParameter("@Password", person.Password));

                con.Open();
                cmd.ExecuteNonQuery();
                con.Close();
            }
        }
        catch (Exception ex)
        {
            throw ex;
        }
    }
}
```

https://www.halvorsen.blog

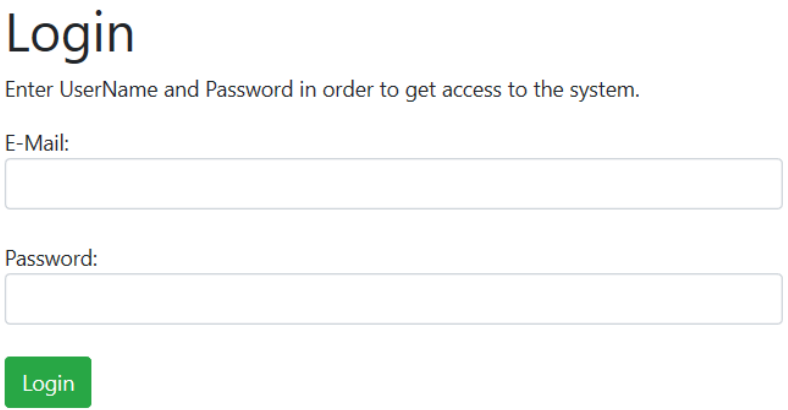
Login

Hans-Petter Halvorsen

Login


LoginApp
Home
Weather
User
Login
Enter UserName and Password in order to get access to the system.
E-Mail:
Password:
Login
Create User
© Developed by Hans-Petter Halvorsen (https://www.halvorsen.blog)

Login



Compare Hashed Password stored in the Database with Password given by User in Login Page

valid = **VerifyHashedPassword**(userName, passwordDB, password);

Code (Login.cshtml.cs)

```
...
public void OnPost()
 {
    CheckLogin();
 }

 void CheckLogin()
 {
    string password;

    userName = Request.Form["EMail"];
    password = Request.Form["Password"];

    Person person = new Person();

    connectionString = _configuration.GetConnectionString("ConnectionString");

    validUser = person.CheckPassword(connectionString, userName, password);

    if (validUser == true)
    {
       HttpContext.Session.SetInt32("Login", 1);
       HttpContext.Session.SetString("UserName", userName);
       Response.Redirect("./User");
    }
    else
    {
       HttpContext.Session.SetInt32("Login", 0);
       HttpContext.Session.SetString("UserName", "");
       Response.Redirect("./Login");
    }

 }
```

Person.cs Class

```
public class Person
{
 ....
 public bool CheckPassword(string connectionString, string userName, string password)
     {
        string passwordDB = null;

        bool loggedIn = false;

        SqlConnection con = new SqlConnection(connectionString);

        string selectSQL = "select Password from PERSON where EMail='" + userName + "'";

        con.Open();

        SqlCommand cmd = new SqlCommand(selectSQL, con);

        SqlDataReader dr = cmd.ExecuteReader();

        if (dr != null)
        {
           while (dr.Read())
           {
              passwordDB = dr["Password"].ToString();
           }
        }

        PasswordHasher<string> pw = new PasswordHasher<string>();

        var verificationResult = pw.VerifyHashedPassword(userName, passwordDB, password);

        if (verificationResult == PasswordVerificationResult.Success)
           loggedIn = true;
        else
           loggedIn = false;

        return loggedIn;
     }
}
```

Web Page (Login.cshtml)

```
<h1>Login</h1>

  <form method="post">

      E-Mail:
      <br />
      <input name="EMail" type="email" class="form-control input-lg" required autofocus />
      <br />

      Password:
      <br />
      <input name="Password" type="password" class="form-control input-lg" required />
      <br />

      <input id="LoginButton" type="submit" value="Login" class="btn btn-success">

  </form>
```

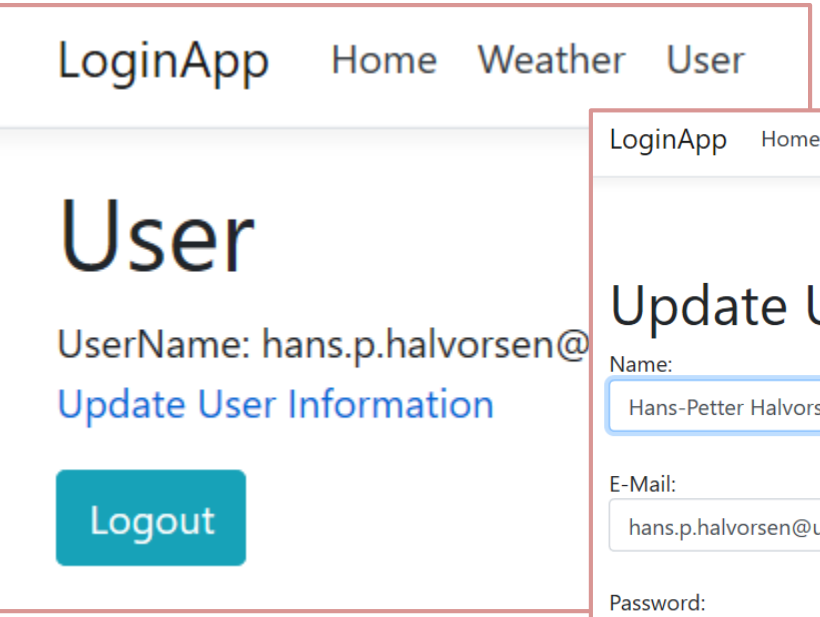
https://www.halvorsen.blog

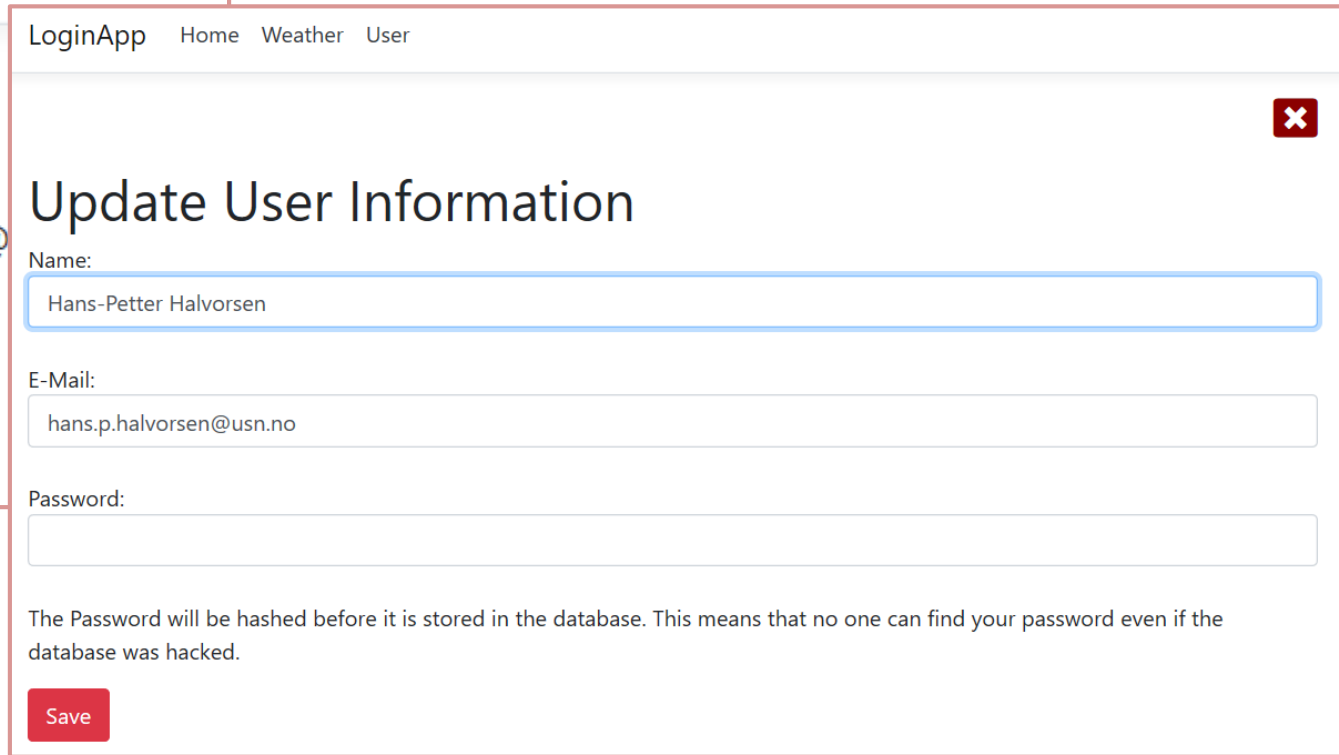
Change User Information

Hans-Petter Halvorsen

Update User Information

Available only when user is logged in





UpdateUser.cshtml.cs

```
 public void OnGet()
{
        VerifyLogin();

        GetUserInformation();
}

public void OnPost()
{
        VerifyLogin();

        UpdateUser();
}
```

Person.cs Class

```
public Person GetUserData(string connectionString, string userName)
{
        Person person = new Person();

        SqlConnection con = new SqlConnection(connectionString);

        string selectSQL = "select PersonId, FullName, EMail from PERSON where EMail='" + userName + "'";

        con.Open();

        SqlCommand cmd = new SqlCommand(selectSQL, con);

        SqlDataReader dr = cmd.ExecuteReader();

        if (dr != null)
        {
           while (dr.Read())
           {
              person.PersonId = Convert.ToInt32(dr["PersonId"]);
              person.FullName = dr["FullName"].ToString();
              person.EMail = dr["EMail"].ToString();
           }
        }

        return person;
}
```

```
public void UpdateUser(string connectionString, Person person)
     {
        try
        {
           using (SqlConnection con = new SqlConnection(connectionString))
           {
              SqlCommand cmd = new SqlCommand("UpdateUser", con);
              cmd.CommandType = CommandType.StoredProcedure;

              cmd.Parameters.Add(new SqlParameter("@PersonId", person.PersonId));
              cmd.Parameters.Add(new SqlParameter("@FullName", person.FullName));
              cmd.Parameters.Add(new SqlParameter("@EMail", person.EMail));
              cmd.Parameters.Add(new SqlParameter("@Password", person.Password));

              con.Open();
              cmd.ExecuteNonQuery();
              con.Close();
           }

        }
        catch (Exception ex)
        {
           throw ex;
        }
     }
```

(UpdateUser.cshtml) Web Page

```
<div>
    <h1>Update User Information</h1>

    <form method="post">

        <input name="PersonId" type="text" value="@Model.personDB.PersonId" hidden />

        Name:
        <br />
        <input name="FullName" type="text" class="form-control input-lg" value="@Model.personDB.FullName" required autofocus />
        <br />

        E-Mail:
        <br />
        <input name="EMail" type="email" class="form-control input-lg" value="@Model.personDB.EMail" required />
        <br />

        Password:
        <br />
        <input name="Password" type="password" class="form-control input-lg" required />
        <br />

        <input id="SaveButton" type="submit" value="Save" class="btn btn-danger">

    </form>

</div>
```

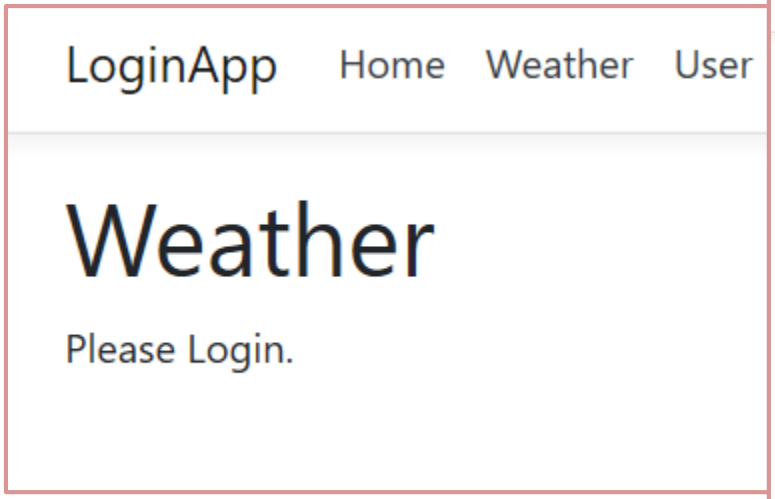
https://www.halvorsen.blog

Weather Data

Hans-Petter Halvorsen

Weather Data

Available only when user is logged in



LoginApp Home Weather User

Weather

UserName: hans.p.halvorsen@usn.no

Below you see Weather Data

Parameter	Value	Unit	TimeStamp
Temperature	22	°C	2020-03-04 11:01:53
Temperature	21	°C	2020-03-04 11:01:53
Temperature	23	°C	2020-03-04 11:01:53
Temperature	19	°C	2020-03-04 11:01:53
Temperature	18	°C	2020-03-04 11:01:53
Temperature	20	°C	2020-03-04 11:01:53
Temperature	24	°C	2020-03-04 11:01:53

Code (Weather.cshtml.cs)

```
public void OnGet()
{
        VerifyLogin();

        WeatherData();
}

 void VerifyLogin()
 {
        if (HttpContext.Session.GetInt32("Login") != null)
           validUser = (int)HttpContext.Session.GetInt32("Login");
        else
           validUser = 0;

        if (HttpContext.Session.GetString("UserName") != null)
           userName = HttpContext.Session.GetString("UserName");
        else
           userName = null;
     }

void WeatherData()
 {
        connectionString = _configuration.GetConnectionString("ConnectionString");

        Weather weather = new Weather();

        weatherData = weather.GetWeatherData(connectionString);
 }
```

Weather.cs Class

```
public class Weather
  {
    public int WeatherParameterId { get; set; }
    public string WeatherParameterName { get; set; }
    public double WeatherParameterValue { get; set; }
    public string WeatherParameterUnit { get; set; }
    public DateTime WeatherParameterTimeStamp { get; set; }

    public List<Weather> GetWeatherData(string connectionString)
    {

      List<Weather> weatherData = new List<Weather>();

      SqlConnection con = new SqlConnection(connectionString);

      string selectSQL = "select WeatherParameterId, ParameterName, [Value], Unit, [TimeStamp] from WeatherData order by [TimeStamp]";

      con.Open();

      SqlCommand cmd = new SqlCommand(selectSQL, con);

      SqlDataReader dr = cmd.ExecuteReader();

      if (dr != null)
      {
        while (dr.Read())
        {
          Weather weather = new Weather();

          weather.WeatherParameterId = Convert.ToInt32(dr["WeatherParameterId"]);
          weather.WeatherParameterName = dr["ParameterName"].ToString();
          weather.WeatherParameterValue = Convert.ToDouble(dr["Value"]);
          weather.WeatherParameterUnit = dr["Unit"].ToString();
          weather.WeatherParameterTimeStamp = Convert.ToDateTime(dr["TimeStamp"]);

          weatherData.Add(weather);
        }
      }
      return weatherData;
    }
  }
```

```
<div>
 <h1>Weather</h1>

 @if (@Model.validUser == 1)
 {
<p>
   <table class="table">
     <thead>
       <tr>
         <th>Parameter</th>
         <th>Value</th>
         <th>Unit</th>
         <th>TimeStamp</th>
       </tr>
     </thead>
     <tbody>
       @foreach (var weather in Model.weatherData)
       {
       <tr>
         <td> @weather.WeatherParameterName </td>
         <td> @weather.WeatherParameterValue </td>
         <td> @weather.WeatherParameterUnit </td>
         <td> @weather.WeatherParameterTimeStamp </td>
       </tr>
       }
     </tbody>
   </table>
</p>
   }
   else
   {
     <p>
       Please Login.
     </p>
   }
</div>
```

Web Page (Weather.cshtml)

https://www.halvorsen.blog

Improvements

Hans-Petter Halvorsen

2 Factor Authentication

- All modern systems offer what we call “2 Factor Authentication”
- This means in addition to enter the password, the user needs to enter a one-time password received on E-Mail or SMS.
- An alternative is to use an Authenticator App like “Google Authenticator” or “Microsoft Authenticator” available on iPhone and Android.

Forgot Password?

- Today we have lots of accounts on different systems
- It is recommended that we use different Passwords for these accounts
- It is the easy to forget the password for one of these accounts
- Because of that we need to have “Forgot Password” functionality
- This means that the user needs to enter his e-mail address and then the system should send and email (or SMS or similar) with a new temporary Password that the user needs to change once he is able to logon to the system again.
- To increase security it is also normal to have some kind of keyword (What is your Nickname?, What is your favorite Pet?, etc.) that the user needs to remember before he can receive a new password.

Resources

- https://docs.microsoft.com/en-us/dotnet/api/microsoft.aspnetcore.identity.passwordhasher-1
- https://stackoverflow.com/questions/40349652/asp-net-core-passwordhashert-issue
- https://docs.microsoft.com/en-us/aspnet/core/fundamentals/app-state
- https://www.learnrazorpages.com/razor-pages/session-state
- https://www.thesslstore.com/blog/difference-encryption-hashing-salting/
- https://auth0.com/blog/adding-salt-to-hashing-a-better-way-to-store-passwords/

Hans-Petter Halvorsen

University of South-Eastern Norway

www.usn.no

E-mail: hans.p.halvorsen@usn.no

Web: https://www.halvorsen.blog