Databasteknik och webbaserade system

Projekt

OtiumActio
Interaktiv aktivitets platform

Niloufar Rashedi Banafshevaragh

Utförd av: *Datum: 2021-10-26*

Första inlämningen

Kursansvarig:
Övriga lärare:

- *Godkänd*
- *Retur*

Rättad av: *Datum:*

Innehållsförteckning

Inledning

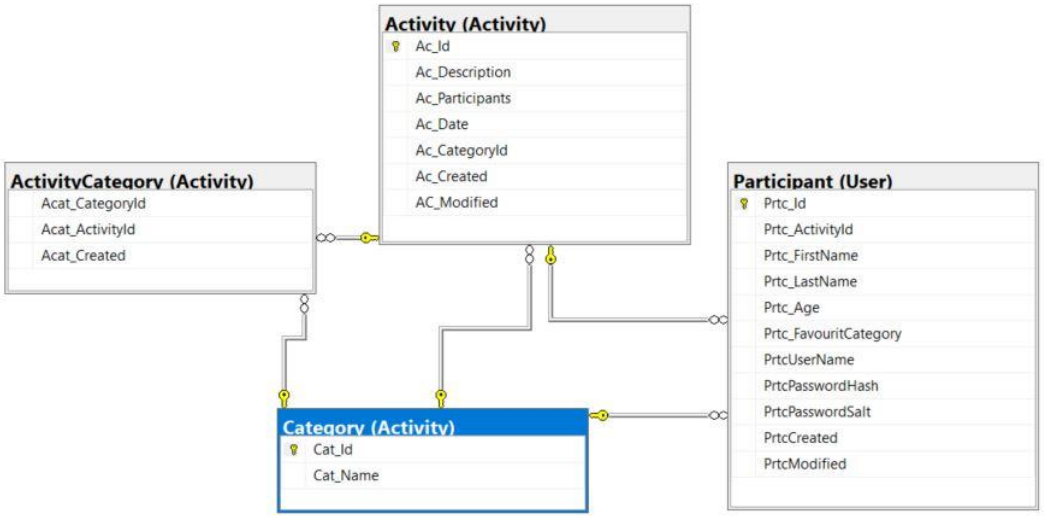
Detta projekt handlar om utveckling av en webbapplikation, OtiumActio, vars grundläggande funktioner utvecklades under Laboration 1 - 3. OtiumActio är ett växande nätverk för gemensamma aktiviteter som man kan välja emelan eller föreslå sina egna aktiviteter. Enligt Projektspecifikation kommer användaren av detta system att utföra fullständiga CRUD operationer. System erbjuder in- och utloggning möjligheter såväl en tjänst som skickar e-mejl för att förnya det glömda lösenordet. Data sparas när man surfar från en sida till en annan i form av Sessions. Syftet med projektet är att visa kunskaper som har inhämtat under kursen. Följande sektioner är detaljerad beskrivning av projektet.

1 Problembeskrivning

Målet av projektet är att utveckla OtiumActio på sättet att projektdirektiv uppfylls. Med hänsyn till CURD operationer som bör finnas i systemet använde jag mig av den befintliga databasen som byggdes upp i föregående laborationer. Utmaningen var att lägga till inloggningsfunktion i systemet och följaktligen nya kolumner som hanterar lösenord i Participant tabell. Vidare fick inmatningsformulär för nya aktiviteter anpassa sig till WCAG och WAI:s rekommendationer. Den genererade HTML koden som härstammade från Razor vyer analyserades enligt W3C:s rekommendationer. Användning av metataggar och att spara information i olika vyer var också ett ytterligare krav som jag försökte uppfylla under projektet. Koden till OtiumActio finns i GitHub under länken https://github.com/niloufar-rashedi/OtiumActio .

2 Systembeskrivning

ER-schema med entiteter (Figur 1) avviker Projektspecifikation i några kolumner, fast den behåller samma relation som presenterades förr. Anledning till nya kolumner i Participant tabell var PasswordSalt och PasswordHash egenskaper som möjliggör omvandling av lösenordet som skickas från vyn med ParticipantDto till säkrare och oläsbar värde. Att förverkliga relationen mellan Participant och Category tabeller i applikationen genom att ha minst en favorit kategori per användare är en del som kommer att hanteras i framtiden.



Figur 1. Relationer mellan tabeller i OtiumActio databasen, hämtad från SQL Server Management Studio 2019, Database Diagrams

OtiumActio lösning förenklades till en Webbaplikation project, .NET Core 3.1, som följs med en bibliotek EmailService där logiken och modeller för att skicka meddelande om det glömde lösenordet genom ett gmail konto lagras.

3 Implementation och gränssnitt

Visual Studio 2019 och SQL Server Manager 18 har varit huvudverktyg som användes sig av för att utveckla projektet. Dessutom använde jag några nuget paket: .NETCore.MailKit 2.0.3 i EmailService [1]; MicrosoftEntityFrameworkCore.Sql/Designer/Tools för att köra migrerationer av den befintliga modeller/relationer och skapa databasen. System.Security.Claims hjälpte också till att skapa Claims vid registrering/inloggning av användare. Applikationen kör på IIS Express.

CRUD operationer, förutom Add/UpdateNewActivity endpoint, hanteras i en generisk repository som ärvs av kontroller. LINQ frågor genomför operationer oberoende av objektet, T. Add/UpdateNewActivity och ResetPassword är basreade på lagrade procedurer som presenterades i laboration 2 och 3.

```
public interface IRepository<T> where T : class
{
        void Add(T entity);
        void Delete(object id);
        void Update(T entity);
        //IQueryable List (Expression<Func<T, bool>> expression);
        IEnumerable<T> GetAll();
        T GetById(object id);
        void Save();
        string AddNewActivity(Activity activity);
        string UpdateActivity(Activity activity);
}
```

Särskilda metoder gällande inloggning och registrering av användare sker genom UserHandler:

```
public interface IUserHandler
{
        void CreatePasswordHash(string password, out byte[] passwordHash, out byte[]
passwordSalt);
        Participant Login(ParticipantLoginDto dto);
        Participant Register(ParticipantDto dto);
        bool VerifyPasswordHash(string password, byte[] storedHash, byte[] storedSalt);
        Participant FindByEmail(string userName);
        string GenerateToken(Participant user);
        Participant UpdatePassword(ResetPasswordDto dto);
}
```

Inloggning är en enkel mechanism där användaren registrerar ett nytt konto i applikationen. I detta steg valideras värde i inmatningsfält. För att logga in med registrerade användarnamn, dvs e-post, fick jag hjälp av HttpContext metoden

som möjliggör SignIn och System.Security.Claim som genererar Claim och ClaimIdentity.

```
[HttpPost]
public async Task<IActionResult> Authenticate(ParticipantLoginDto dto)
{
if (!ModelState.IsValid)
{
return View("Index");
}
else
{
try
{
        var user = _handler.Login(dto);
        if (user == null)
        {
        ModelState.AddModelError("", "Användernamn el. lösenord är fel");
        return View("LoginForm");

        }
        var claims = new List<Claim>() {
        new Claim(ClaimTypes.NameIdentifier, Convert.ToString(user.PrtcId)),
        new Claim(ClaimTypes.Name, user.PrtcFirstName)

        };
        var identity = new ClaimsIdentity(claims,
        CookieAuthenticationDefaults.AuthenticationScheme);
        var principal = new ClaimsPrincipal(identity);
        await
        HttpContext.SignInAsync(CookieAuthenticationDefaults.AuthenticationSc
        heme, principal, new AuthenticationProperties());
```

EmailSender tjänsten skickar länken till användarens e-post som hämtas från Participant tabellen. .NETCore.MailKit paket, baserad på SMTP, skapar det nya meddelandet och läser upp uppgifter som e-post vilken meddelanden skickar ifrån , smtp port, lösenord, … från appsettings.json filen. Det skickas ivär från Smtp servern smtp.gmail.com och slutligen får användaren länken i sin mejlkorg.

```
private MimeMessage CreateEmailMessage(Message message)
{
var emailMessage = new MimeMessage();
emailMessage.From.Add(new MailboxAddress(_emailConfig.From));
emailMessage.To.AddRange(message.To);
emailMessage.Subject = message.Subject;
emailMessage.Body = new TextPart(MimeKit.Text.TextFormat.Html) { Text =
string.Format("<h2 style='color:red;'>{0}</h2>", message.Content) };
return emailMessage;
}
…
private void Send(MimeMessage mailMessage)
{
using (var client = new SmtpClient())
{
```

```
        try
        {
                client.ServerCertificateValidationCallback = (s, c, h, e) => true;
                client.Connect(_emailConfig.SmtpServer, _emailConfig.Port, true);
                client.AuthenticationMechanisms.Remove("XOAUTH2");
                client.Authenticate(_emailConfig.UserName, _emailConfig.Password);
                client.Send(mailMessage);
                }
                catch(Exception ex)
                {
                        //log an error message or throw an exception or both.
                        throw new Exception(ex.Message);
                }
                finally
                {
                        client.Disconnect(true);
                        client.Dispose();
                }
                }
                }
```

…

```
        {
         "AppSettings": {
                "Secret": "SuperSecretPasswordOtiumActioThatMustBeHiddenSomeWhere"
         },

         "EmailConfiguration": {
                "From": "otiumactio@gmail.com",
                "SmtpServer": "smtp.gmail.com",
                "Port": 465,
                "Username": "otiumactio@gmail.com",
                "Password": "*******"
         },
```

Metoden i UserController där äntligen callback och meddelande skapas:

```
        [HttpPost]
        [ValidateAntiForgeryToken]
        public IActionResult ForgotPassword(ForgottenPasswordDto dto)
        {
        if (!ModelState.IsValid)
        {
        ModelState.AddModelError("", "E-posten är ogiltig");
        return View("EnterEmail");
        }
         var user = _handler.FindByEmail(dto.PrtcUserName);

        if (user == null)
        return RedirectToAction(nameof(ForgotPasswordConfirmation));
         var token = _handler.GenerateToken(user);

        var callback = Url.Action(nameof(ResetPassword), "User", new { token, email =
        user.PrtcUserName }, Request.Scheme);
        var message = new Message(new string[] { user.PrtcUserName }, "Reset password
token", callback, null);
        _emailSender.SendEmail(message);
```

```
return RedirectToAction(nameof(ForgotPasswordConfirmation));
}
```

Förutom att logga in i applikationen har inloggningssytem ett annat syfte som är att förhindra oautentiserade användare att skapa ny aktivitet. Därför SuggestedActivityController märkerades med [Authorize] attribut.

Ett exempel där session sparar data är när användaren tar bort en aktivitet från aktivitetslistan ochi slutet av metoden skapas session och efteråt TempData som kommer att dycka upp i huvudmenyn i alla vyer:

```
public IActionResult Delete(Activity activity)
{
var deletecActivity = _context.Activities.Where(a => a.AcId == activity.AcId).Select(a
=> a.AcDescription).FirstOrDefault();
_service.Delete(activity.AcId);
var sessionValue = $"Du har nyss tagit bort aktiviteten |{deletecActivity}|!";
var sessionName = "SuccessDelete";
HttpContext.Session.SetString(sessionName, sessionValue);
TempData["DeletedActivity"] = HttpContext.Session.GetString(sessionName);
TempData.Keep();
return RedirectToAction("Index", "Home");
}
```

Och i _Layout vyn:

…

```
@if (TempData["DeletedActivity"] != null)
{
<li class="nav-item">
<p><i>@TempData["DeletedActivity"].ToString()</i></p>
</li>

}
```

Inmatningsformulären skapades enligt WCAG och WAI:s rekommendationer och har motsvarande HTML taggar som markerar olika delar av formulären [1]. Index.cshtml som ligger under SuggestedActivity mappen i vyer är ett exempel på detta:

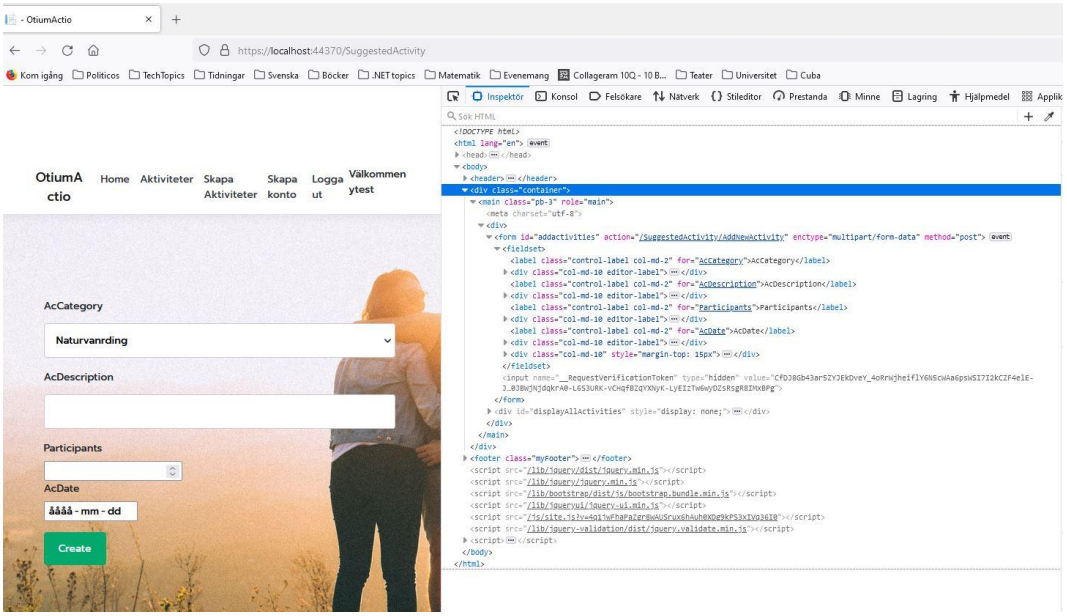
```
@model OtiumActio.Models.Activity
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8" />
</head>
<body>
    <div>
    @using (Html.BeginForm("AddNewActivity", "SuggestedActivity", FormMethod.Post,
new { id = "addactivities", enctype = "multipart/form-data" }))
    {
    @Html.ValidationSummary(true, "", new { @class = "text-danger" })
    <fieldset>
```

```
            @Html.LabelFor(model => model.AcCategory, new { @class = "control-label
col-md-2" })
            <div class="col-md-10 editor-label">
                        @Html.DropDownList("AcCategoryId", (IEnumerable
            <SelectListItem>)ViewData["SelectableCategories"])

            </div>
            @Html.LabelFor(model => model.AcDescription, new { @class = "control-label
col-md-2" })
            <div class="col-md-10 editor-label">
                        @Html.EditorFor(model => model.AcDescription)
                        @Html.ValidationMessageFor(model => model.AcDescription, "", new {
@class = "text-danger" })
            </div>
            @Html.LabelFor(model => model.Participants, new { @class = "control-label col-md-2"
})
            <div class="col-md-10 editor-label">
                        @Html.EditorFor(model => model.AcParticipants)
                        @Html.ValidationMessageFor(model => model.AcParticipants, "", new {
@class = "text-danger" })
            </div>
            @Html.LabelFor(model => model.AcDate, new { @class = "control-label col-md-2" })
            <div class="col-md-10 editor-label">
                        @Html.EditorFor(model => model.AcDate, new { @class = "date" })
            </div>
            <div class="col-md-10" style="margin-top: 15px">
                        <input type="submit" value="Create" class="btn btn-default" />
            </div>
            </fieldset>
            }
            <div id="displayAllActivities" style="display: none;">
            <p>Tack för att förslå en ny aktivitet!</p>
            </div>

            </div>
            </body>
</html>
            @section scripts
            {
            <script src="~/lib/jquery-validation/dist/jquery.validate.min.js"></script>
            <script>
            $('#addactivities').submit(function () {
            $('#displayAllActivities').show()
            });
            </script>
            }
```

Den ovanstående koden tolkas till rent HTML kod i webbläsaren, Firefox 93.0 (64-bitars). Figur 2 visar den genererade koden.



Figur 2. den genererade HTML koden i Firefox webbläsare (inmatningsformulär för nya aktiviteter)

I samband med punkt 5 i projektdirektiv har huvudvyn, dvs hemmsida, aktivitessida och också "skapa ny aktivitet" sida med lämpliga metataggar [3] som lättar eventuella sökningar på nätet för att leda användaren till OtiumActio. Här nedan listas några av dem:

- I Home/Index.cshtml vy

```
@model IEnumerable<OtiumActio.Models.Category>
@{
        ViewData["Title"] = "Otium Actio";
}
<!DOCTYPE html>
<html>

<head>
        <meta name="keywords" content="Sociala aktiviteter, Matlöagning, Föreläsning, Bio, Naturvandring" />
        <meta name="description" content="Sociala aktiviteter som kan skapas eller väljas emelan. OtiumActio är ett latiskt ord för Fritidsaktivitet, ett växande nätverk för gemensamma aktiviteter." />
</head>

<body>
        <div> ...
```

- I Activity/Index.cshtml vy där skapade aktiviteter listas och användaren omdirigeras till "skapa ny aktivitet" sida efter 5 sekunder från databasen. Tanken var att hamna i en annan sida utan att URL:en ändras och utan att användaren känner sig att vara i en helt annan sida. Var OtiumActio e-handel applikation skulle ha funnits bättre exempel på denna funktion:

```
@model IEnumerable<OtiumActio.Models.Activity>
<!DOCTYPE html>
<html>
```

```
<head>
        <meta name="keywords" content="Sociala aktiviteter, OtiumActio,
Sverige, Samhälle'" />
        <meta http-equiv="refresh" content="5; url =
https://localhost:44370/SuggestedActivity" />
</head>
<body>
        <div>...
```

4 Lösningens begränsningar

OtiumActio kör fortfarande på användarens nivå. Alltså finns det ingen behörighetstilldelning mellan Admin och User. Det skiljer systemet från en "real-world" applikation och det krävs vidare refaktorering och förbättring av system för att närmar sig till verklighet. Vidare använder systemet en blandning av lagrade procedurer och LINQ frågor att hantera CRUD operationer. Det kan enas i framtiden eller kombinerad på ett bättre sätt. I motsats till Laboration 3 där sökning, filtrering och sortering befinner sig i applikation saknas sådana funktioner från projektet. Anledning var prioriteringar i projektdirektiv och faktum att applikationen kommunicerar med databasen inte genom DAL, utan genom en generisk repository som krävde granskning och refaktorering för att få dessa funktioner på plats.

5 Problem och reflektioner

Först och främst var projektet ambitiöst planerat. Med hänsyn till antal deltagare i projektet, dvs bara jag själv, och utmaningar som dök upp förenklades projektet och avvek från vad hävdades i Projektspecifikation. Att förse MVC applikation direkt med IdentityServer 4 tog längre tid och att ha en API lösning mellan IdentityServer och MVC applikation komplicerade projektet onödigt. Därför bestämde jag mig att begränsa autentiseringsprocessen till det enkla systemet som förklarades ovan. Att skicka påminnelser om aktiviteter som användaren har visat intresse för ersattes med en tjänst som skickar länk till lösenords förnyelse. För att förbättra systemet i framtiden behövs det å ena sida att refaktorera koden, särskilt när det gäller kommunikation med databasen, och å andra sida att implementera en mer säkrare och komplicerade autentiseringssystem för att sjösätta applikationen i verklighet.

6 Referenser

[1] CodeMaze (2021) https://code-maze.com/password-reset-aspnet-core-identity/ (Hämtad 2021-10-19).
[2] Web Accessibility Tutorials (2019) https://www.w3.org/WAI/tutorials/forms/ (Hämtad 2021-10-20).
[3] Tutorialspoint (2021) https://www.tutorialspoint.com/html/html_meta_tags.htm (Hämtad 2021-10-24).