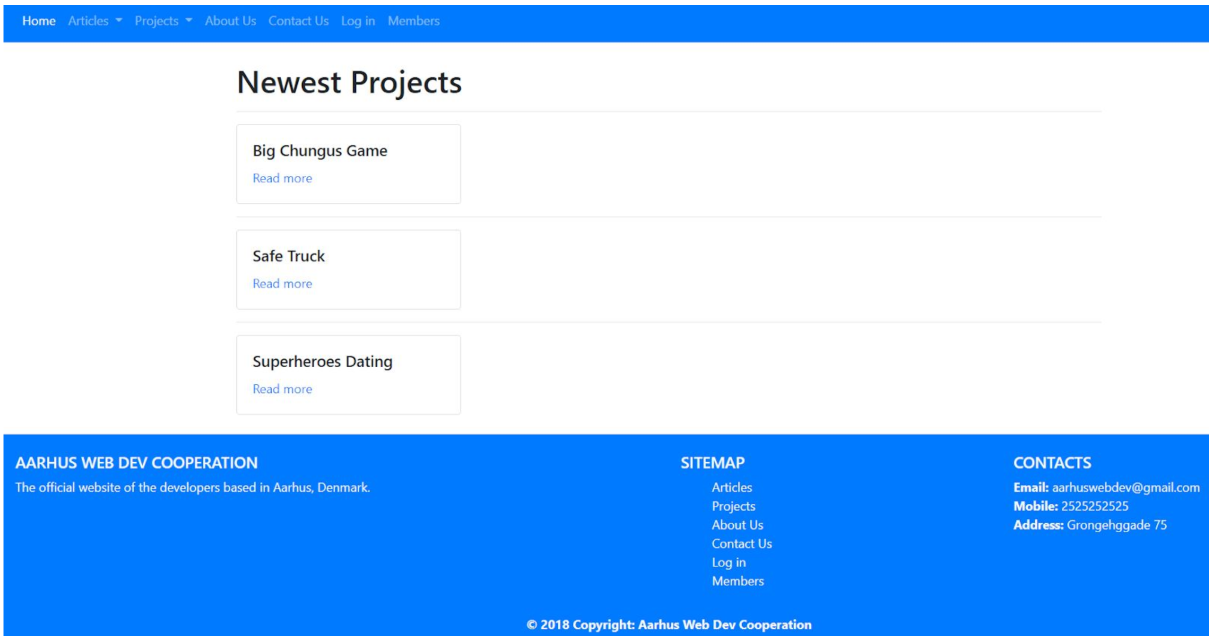
Aarhus Web Dev Cooperation

Mandatory Assignment

Written by : Alexander Kettunen & Georgi Ovalov

https://github.com/Tanngar/AarhusWebDevCoop



Newest Projects

Big Chungus Game
Read more

Safe Truck
Read more

Superheroes Dating
Read more

AARHUS WEB DEV COOPERATION
The official website of the developers based in Aarhus, Denmark.

SITEMAP
Articles
Projects
About Us
Contact Us
Log in
Members

CONTACTS
Email: aarhuswebdev@gmail.com
Mobile: 2525252525
Address: Grongehggade 75

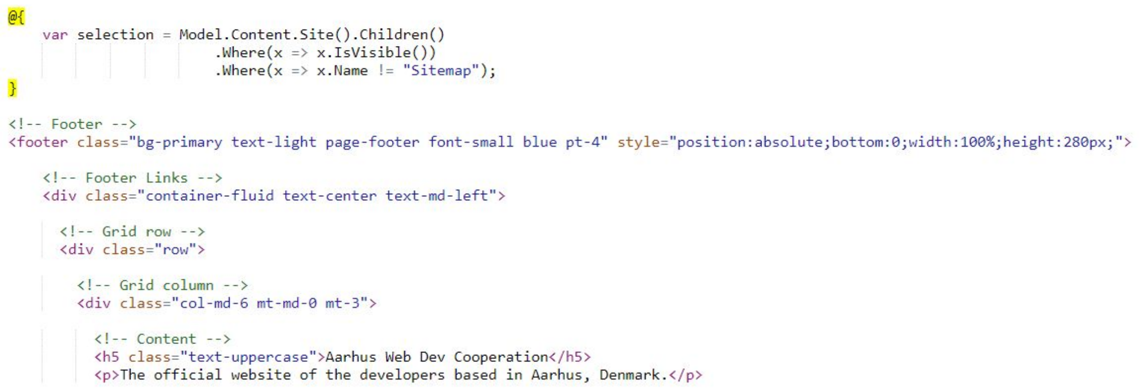
© 2018 Copyright: Aarhus Web Dev Cooperation

Introduction

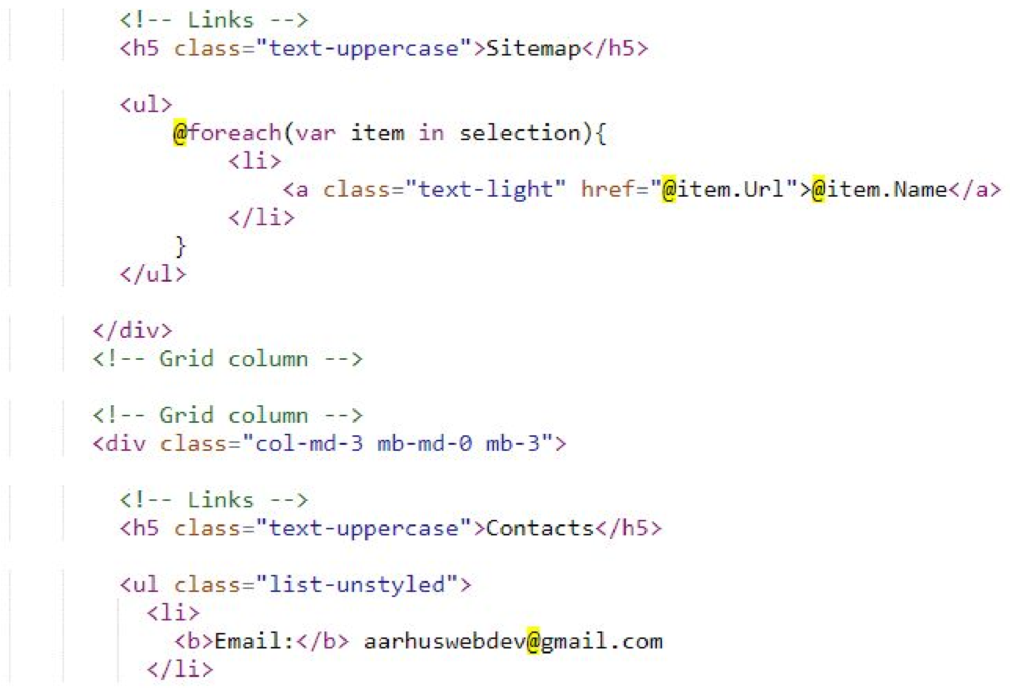
For this mandatory assignment in our Development Environments course, we had to build a website using Umbraco CMS. We had already a set theme for it, which is Aarhus Web Development Cooperation. We were to build all the necessary menus required by our teacher and put some more content that we had to come up with. We used to follow the exercises given in each of our Umbraco lessons in order to get a fully functional website in the end. The finished website would consists of main navigation menu with the following options : “Home” which directs you to the main page, “Articles” which is a drop-down menu with all of the articles published on the website, “Project” option with again a drop-down with all of the projects, which dependent on whether you are logged in or not would either display or not, “About Us” page where you can read more about the organisation itself, their vision and so on, “Contact Us” page with a form, where you can input information in fields and send it, “Log in” where you can login as an user and get permission to see all the projects for example, “Members” which is for the members of this organisation with some additional information about each of them. Then you have a “Site map” with all the possibility pages you can navigate through in the website displayed in the footer together with some more information.

Site map

As we already know from our previous lessons in Web Development education, site maps are a good tool for structuring the pages in hierarchical human visible listing and also for better search engines results. Our sitemap consists of all the possible menus we have in the navigation bar. It appears in our footer (PLACE AN IMAGE HERE). For generating the sitemap, we have created a partial view, which later we have included to the specific template fails where we need it.

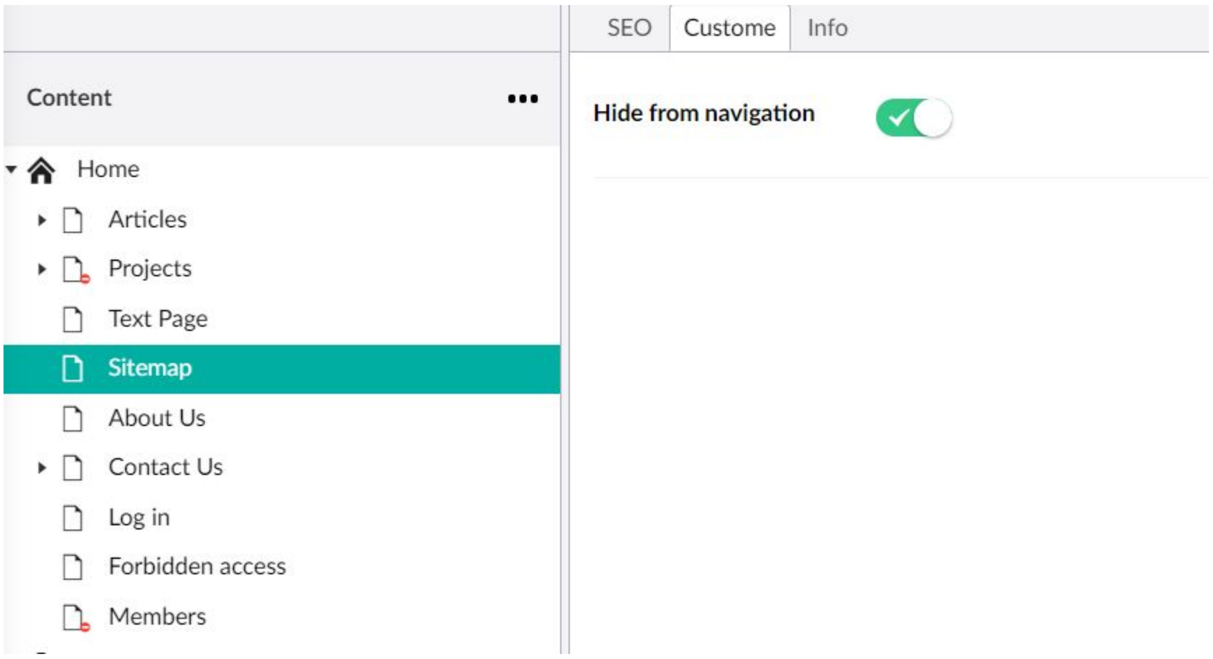


```
@{
    var selection = Model.Content.Site().Children()
                        .Where(x => x.IsVisible())
                        .Where(x => x.Name != "Sitemap");
}

<!-- Footer -->
<footer class="bg-primary text-light page-footer font-small blue pt-4" style="position:absolute;bottom:0;width:100%;height:280px;">

    <!-- Footer Links -->
    <div class="container-fluid text-center text-md-left">

      <!-- Grid row -->
      <div class="row">

        <!-- Grid column -->
        <div class="col-md-6 mt-md-0 mt-3">

          <!-- Content -->
          <h5 class="text-uppercase">Aarhus Web Dev Cooperation</h5>
          <p>The official website of the developers based in Aarhus, Denmark.</p>
```



```
          <!-- Links -->
          <h5 class="text-uppercase">Sitemap</h5>

          <ul>
              @foreach(var item in selection){
                  <li>
                      <a class="text-light" href="@item.Url">@item.Name</a>
                  </li>
              }
          </ul>

        </div>
        <!-- Grid column -->

        <!-- Grid column -->
        <div class="col-md-3 mb-md-0 mb-3">

          <!-- Links -->
          <h5 class="text-uppercase">Contacts</h5>

          <ul class="list-unstyled">
            <li>
              <b>Email:</b> aarhuswebdev@gmail.com
            </li>
```

We have requested to use the sitemap in the footer for maximum realistic look of the website. Another feature we have been able to use is to hide it from the navigation menu, because simply we don’t need it up there.



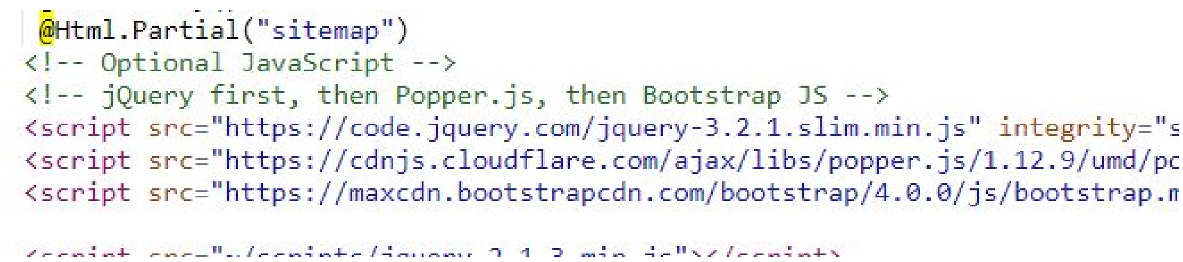
SEO | Custome | Info

Hide from navigation ✓

Content

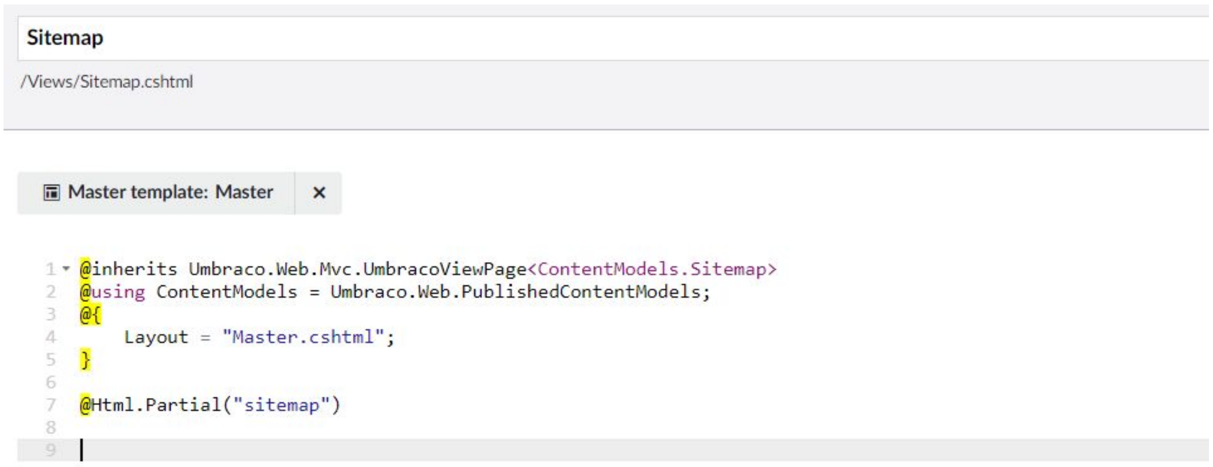
- Home
  - Articles
  - Projects
  - Text Page
  - Sitemap
  - About Us
  - Contact Us
  - Log in
  - Forbidden access
  - Members

This is how he have included it inside the master template :



```
@Html.Partial("sitemap")
<!-- Optional JavaScript -->
<!-- jQuery first, then Popper.js, then Bootstrap JS -->
<script src="https://code.jquery.com/jquery-3.2.1.slim.min.js" integrity="s
<script src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.12.9/umd/pc
<script src="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0/js/bootstrap.m
```

And this is the Sitemap template itself, again we have included it as a partial view and set to inherit from the Master template :



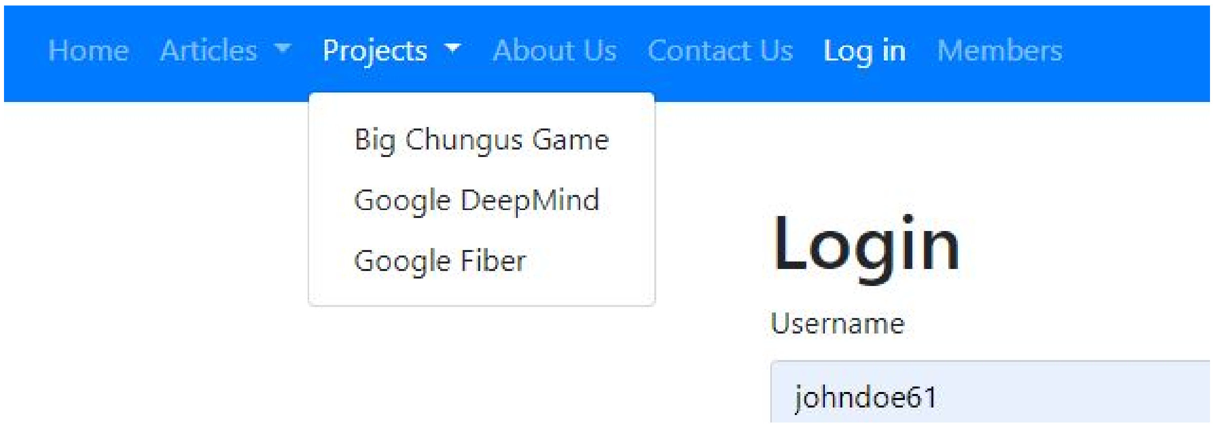
```
@inherits Umbraco.Web.Mvc.UmbracoViewPage<ContentModels.Sitemap>
@using ContentModels = Umbraco.Web.PublishedContentModels;
@{
    Layout = "Master.cshtml";
}

@Html.Partial("sitemap")

```

Website Menu

The code is explained line by line as comments inside the Umbraco backoffice/Visual studio files

Website Access Control

Users that are not logged in don't have access to members page and are only able to see the completed projects in the dropdown:



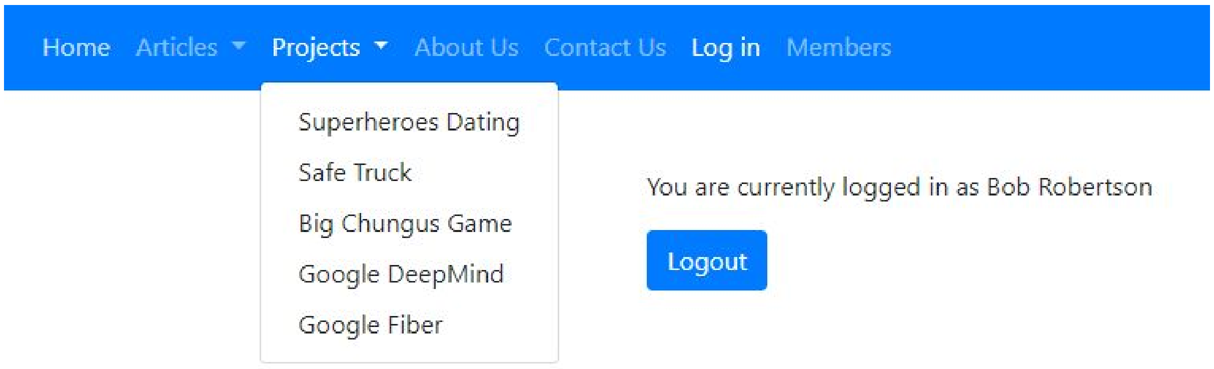
Login

Username

johndoe61

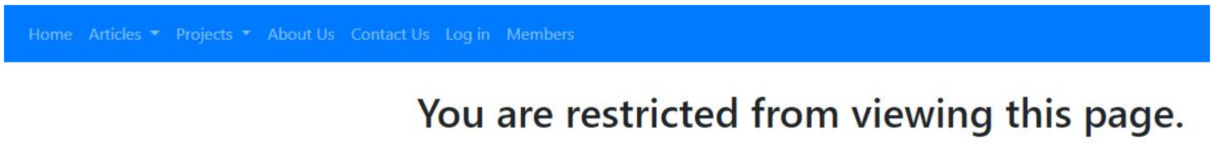
If the user tries to access one of the protected projects when they are not logged in, they would be redirected to the login page.

Users logged in as guest members have access to ‘completed’, ‘initiated’ and ‘under development’ projects:



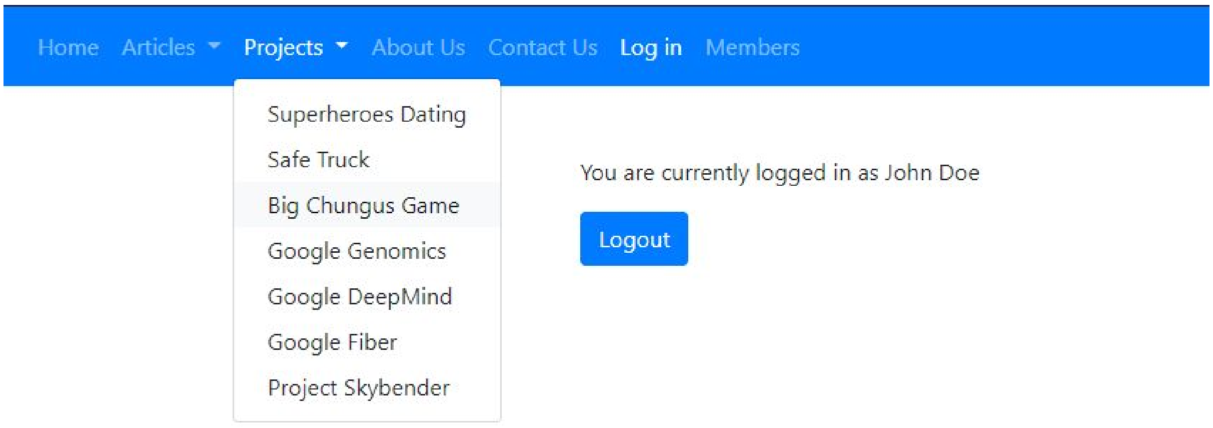
You are currently logged in as Bob Robertson

If a logged in user tries to access a page that he does not have access to, they would be redirected to forbidden access page:



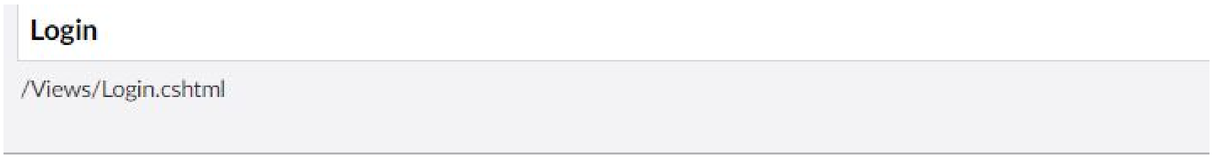
You are restricted from viewing this page.

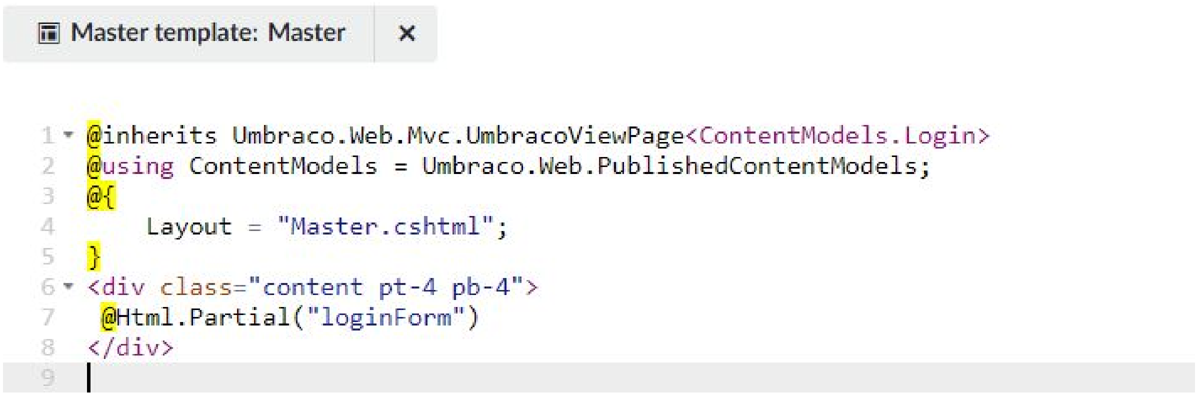
Users logged in as active members have access to all projects:



Login System

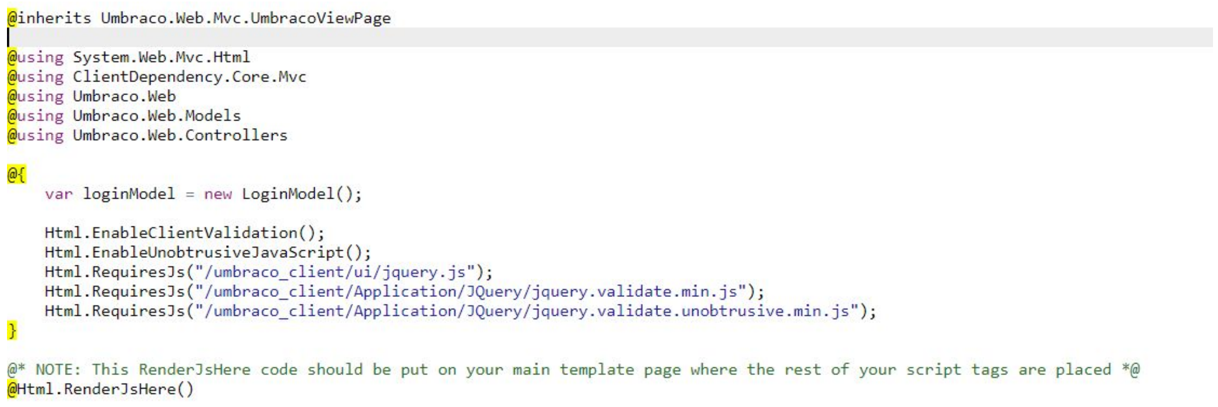
For the login part, we have created an empty document type and a template, which again inherits from the Master





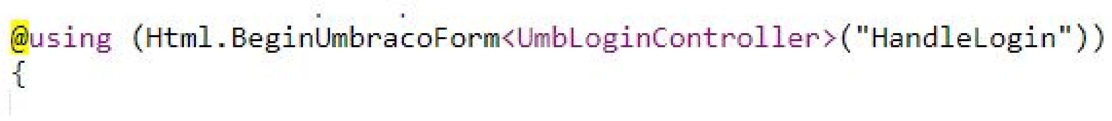
```
@inherits Umbraco.Web.Mvc.UmbracoViewPage<ContentModels.Login>
@using ContentModels = Umbraco.Web.PublishedContentModels;
@{
    Layout = "Master.cshtml";
}
<div class="content pt-4 pb-4">
 @Html.Partial("loginForm")
</div>
```

In order to implement the actual log in, we have created a partial view named loginForm and inside it, the first thing we have stated are the JS packages, which we use to make the client side validation for it.



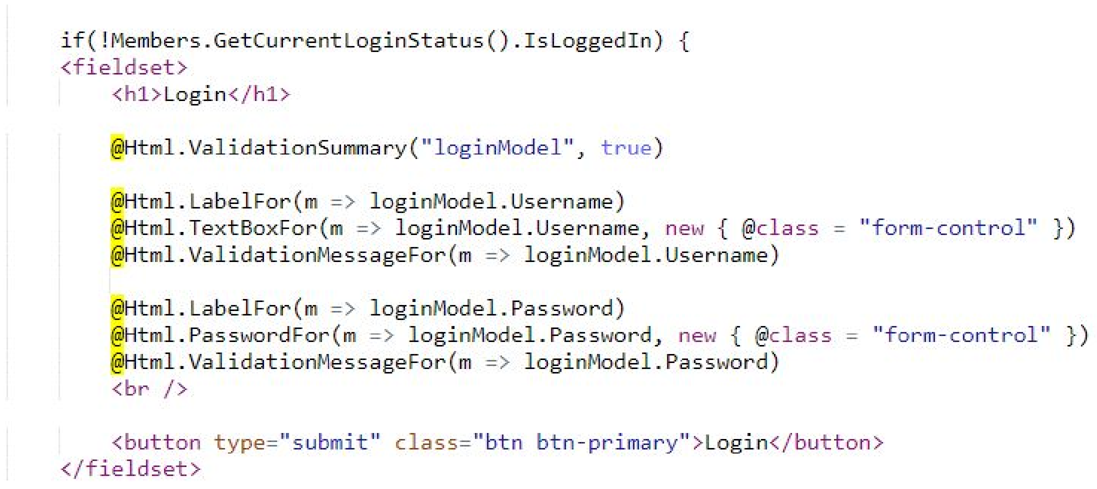
```
@inherits Umbraco.Web.Mvc.UmbracoViewPage

@using System.Web.Mvc.Html
@using ClientDependency.Core.Mvc
@using Umbraco.Web
@using Umbraco.Web.Models
@using Umbraco.Web.Controllers

@{
    var loginModel = new LoginModel();

    Html.EnableClientValidation();
    Html.EnableUnobtrusiveJavaScript();
    Html.RequiresJs("/umbraco_client/ui/jquery.js");
    Html.RequiresJs("/umbraco_client/Application/JQuery/jquery.validate.min.js");
    Html.RequiresJs("/umbraco_client/Application/JQuery/jquery.validate.unobtrusive.min.js");
}

@* NOTE: This RenderJsHere code should be put on your main template page where the rest of your script tags are placed *@
@Html.RenderJsHere()
```

To make it to work, we use :



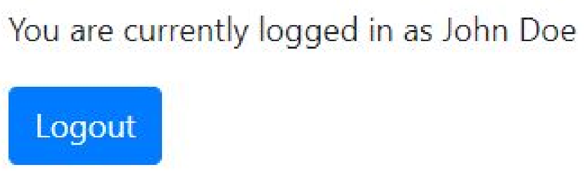
```
@using (Html.BeginUmbracoForm<UmbLoginController>("HandleLogin"))
{
```

a controller, which would handle the login requests.

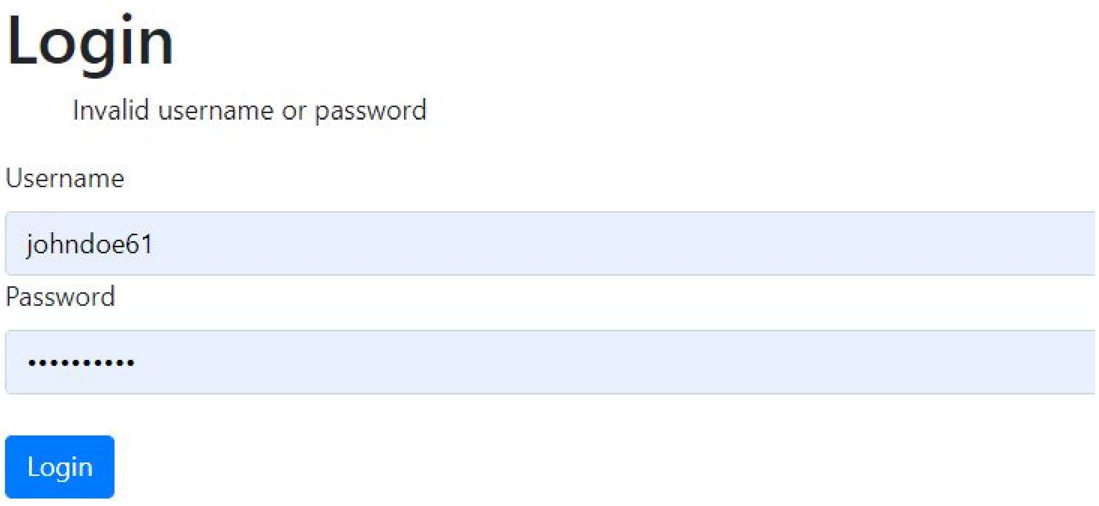


```
    if(!Members.GetCurrentLoginStatus().IsLoggedIn) {
    <fieldset>
        <h1>Login</h1>

        @Html.ValidationSummary("loginModel", true)

        @Html.LabelFor(m => loginModel.Username)
        @Html.TextBoxFor(m => loginModel.Username, new { @class = "form-control" })
        @Html.ValidationMessageFor(m => loginModel.Username)

        @Html.LabelFor(m => loginModel.Password)
        @Html.PasswordFor(m => loginModel.Password, new { @class = "form-control" })
        @Html.ValidationMessageFor(m => loginModel.Password)
        <br />

        <button type="submit" class="btn btn-primary">Login</button>
    </fieldset>
```

Then we have an if statement checking the current log in status, where there is someone logged in or not and depending on this, it would display you either the fields for log in or it will state that you are already a logged in user.
The lines with @Html are using the client-side validation that we included through the JS packages. Whenever you input a true values for user and password you would be successfully redirected to:


You are currently logged in as John Doe

and if you have inputted false values, then it use the client-side validation to display you that either your username or password is wrong:


Login

Invalid username or password

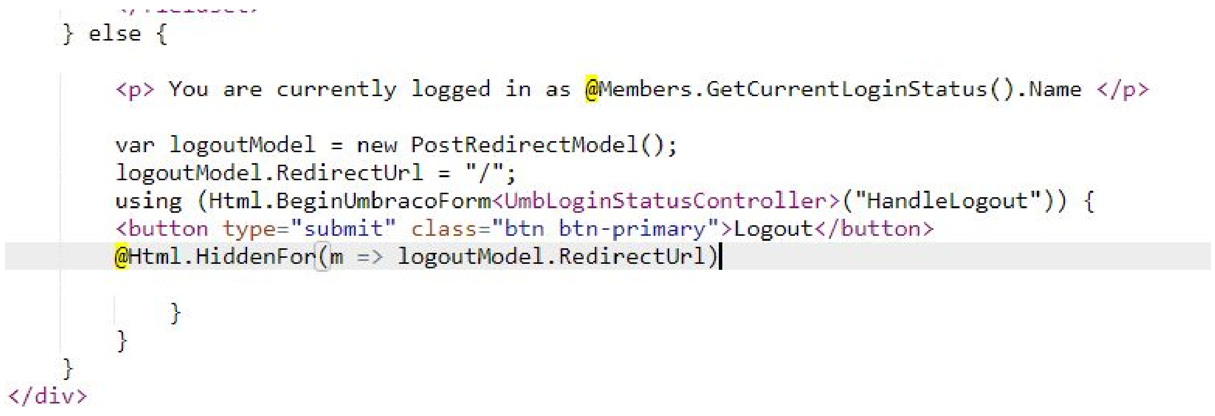
Username

johndoe61

Password

••••••••••

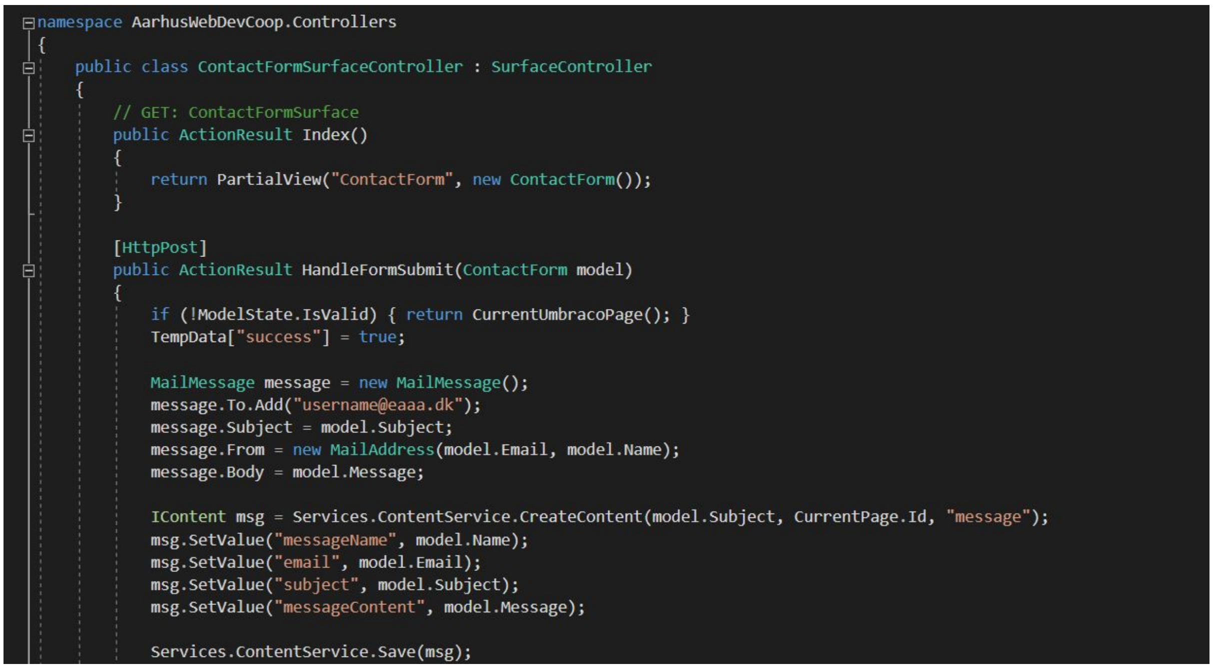
Also we have an if statement, which we showed for the log in part, later in the code we have an else statement:


```
    } else {

        <p> You are currently logged in as @Members.GetCurrentLoginStatus().Name </p>

        var logoutModel = new PostRedirectModel();
        logoutModel.RedirectUrl = "/";
        using (Html.BeginUmbracoForm<UmbLoginStatusController>("HandleLogout")) {
        <button type="submit" class="btn btn-primary">Logout</button>
        @Html.HiddenFor(m => logoutModel.RedirectUrl)

            }
        }
    }
</div>
```

which is used to handle the log out requests we make. All the members are allowed to log in with their credentials, but what is different among them is the level of access they have around the website, said in other words - what are they allowed to see and what are they not allowed to see.

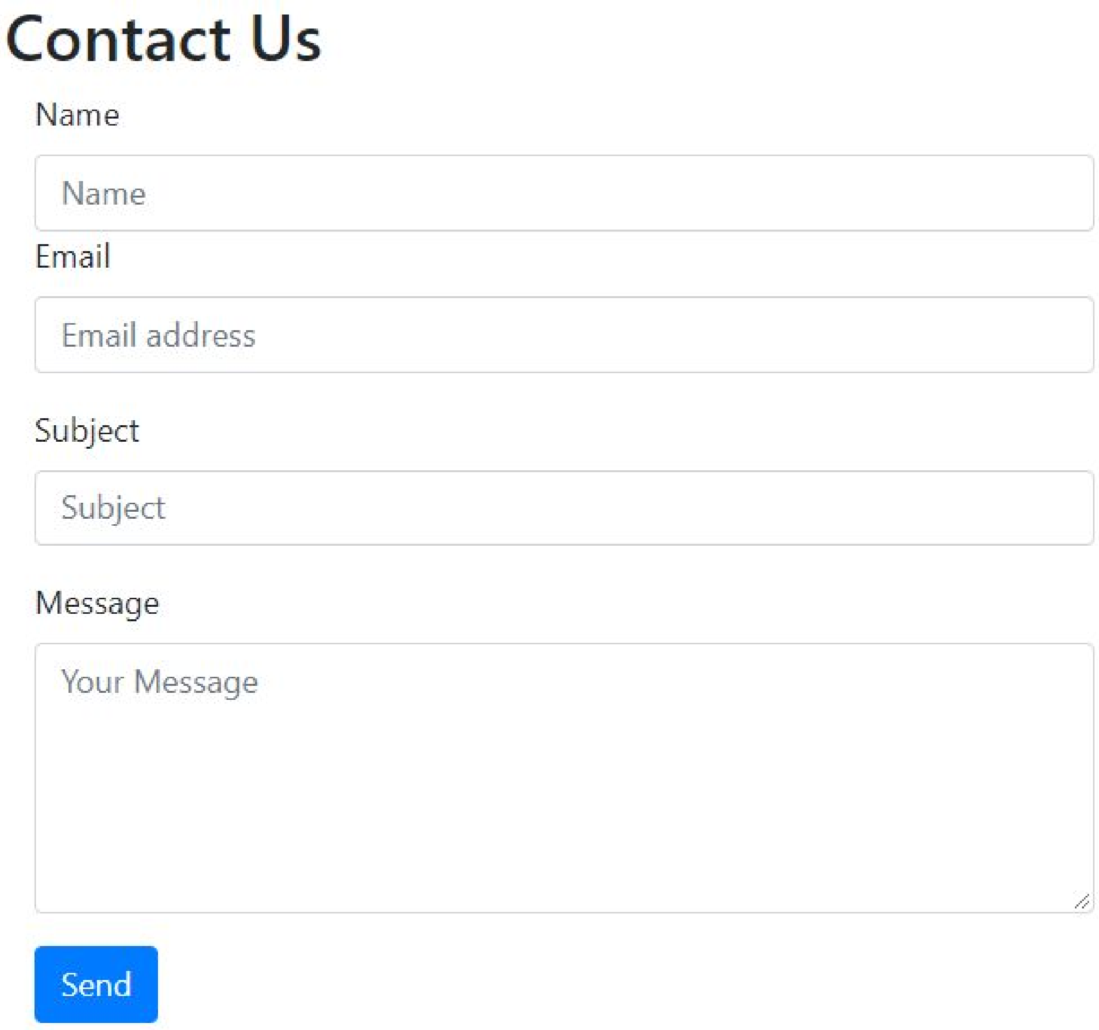
Contact form with Surface Controller

Worth mentioning at first is that without using the Surface Controller it is not possible to have any interaction in the website, which means that we cannot have any forms to fill out, log in or just simply any kind of user input. They can be called in two ways, through templates and through partial views.
In the first part of the code inside the SurfaceController

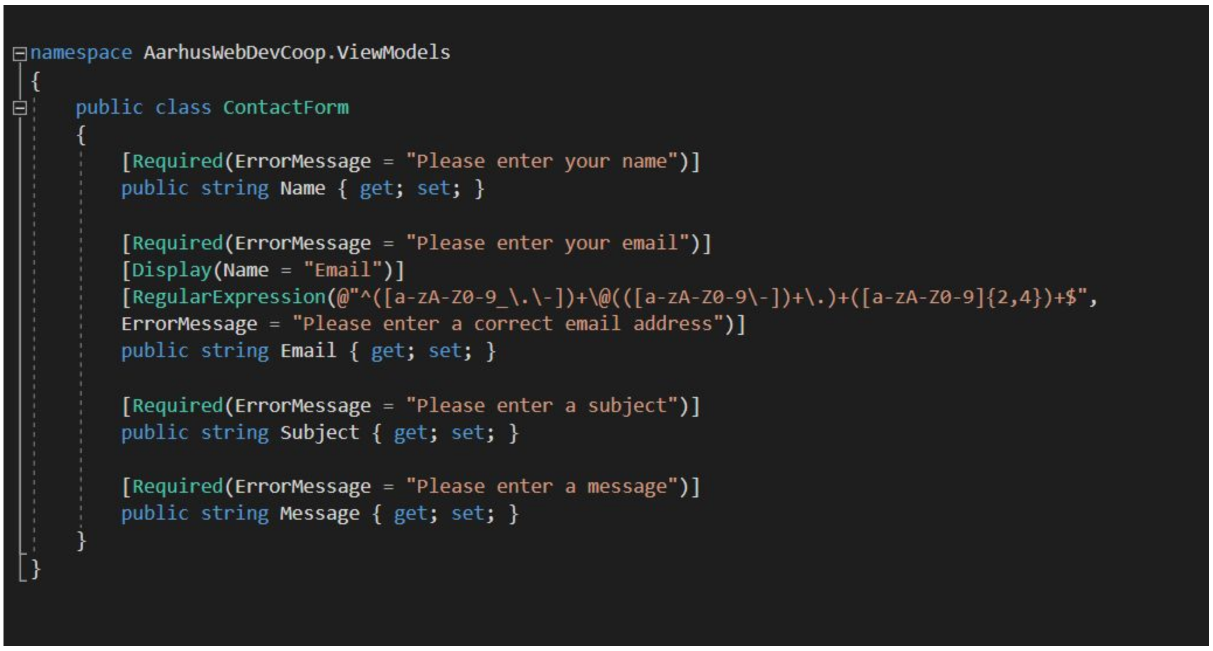


```
namespace AarhusWebDevCoop.Controllers
{
    public class ContactFormSurfaceController : SurfaceController
    {
        // GET: ContactFormSurface
        public ActionResult Index()
        {
            return PartialView("ContactForm", new ContactForm());
        }

        [HttpPost]
        public ActionResult HandleFormSubmit(ContactForm model)
        {
            if (!ModelState.IsValid) { return CurrentUmbracoPage(); }
            TempData["success"] = true;

            MailMessage message = new MailMessage();
            message.To.Add("username@eaaa.dk");
            message.Subject = model.Subject;
            message.From = new MailAddress(model.Email, model.Name);
            message.Body = model.Message;

            IContent msg = Services.ContentService.CreateContent(model.Subject, CurrentPage.Id, "message");
            msg.SetValue("messageName", model.Name);
            msg.SetValue("email", model.Email);
            msg.SetValue("subject", model.Subject);
            msg.SetValue("messageContent", model.Message);

            Services.ContentService.Save(msg);
```

We have an ActionResult method, which returns the ContactForm partial view with the fields to be filled out :

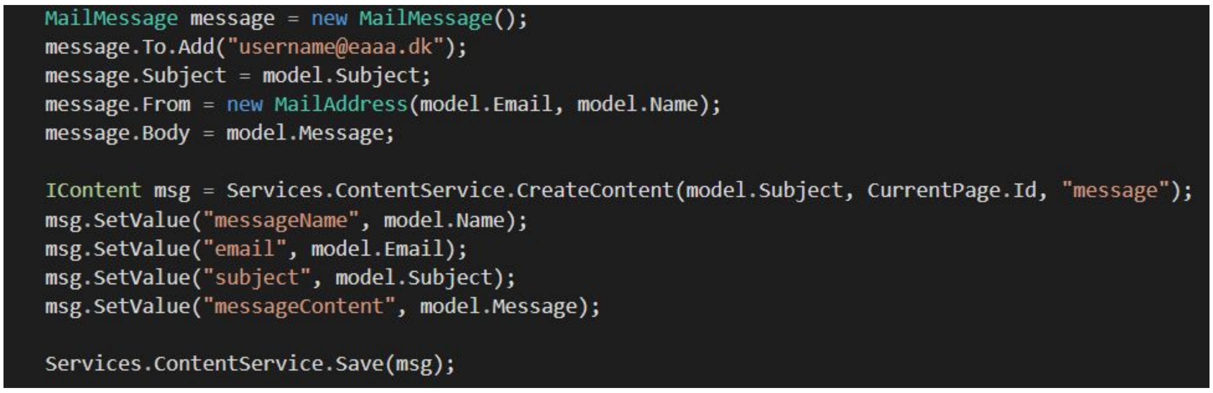


It takes it from the folder ViewModels, which contains the ContactForm with the following code:



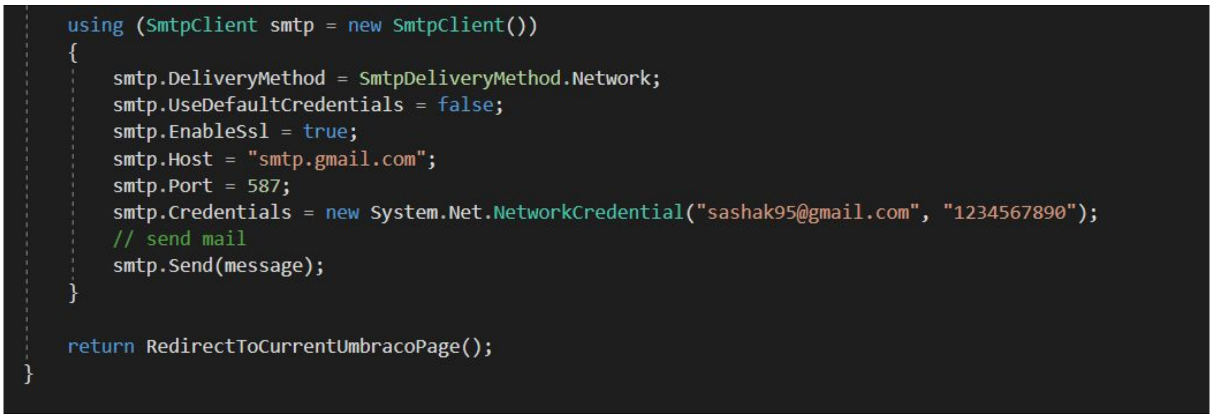
```
namespace AarhusWebDevCoop.ViewModels
{
    public class ContactForm
    {
        [Required(ErrorMessage = "Please enter your name")]
        public string Name { get; set; }

        [Required(ErrorMessage = "Please enter your email")]
        [Display(Name = "Email")]
        [RegularExpression(@"^([a-zA-Z0-9_\.\-])+\@(([a-zA-Z0-9\-])+\.)+([a-zA-Z0-9]{2,4})+$",
        ErrorMessage = "Please enter a correct email address")]
        public string Email { get; set; }

        [Required(ErrorMessage = "Please enter a subject")]
        public string Subject { get; set; }

        [Required(ErrorMessage = "Please enter a message")]
        public string Message { get; set; }
    }
}
```

We can see the declared properties that the class consists of. We have also DataAnnotation for client-side validation for each of the fields.
Then later in the SurfaceController we have an if statement, which checks whether we have inputted data that would be accepted by the validation and if yes, we assign the values to the properties of the ContactForm class and then save it.



```
MailMessage message = new MailMessage();
message.To.Add("username@eaaa.dk");
message.Subject = model.Subject;
message.From = new MailAddress(model.Email, model.Name);
message.Body = model.Message;

IContent msg = Services.ContentService.CreateContent(model.Subject, CurrentPage.Id, "message");
msg.SetValue("messageName", model.Name);
msg.SetValue("email", model.Email);
msg.SetValue("subject", model.Subject);
msg.SetValue("messageContent", model.Message);

Services.ContentService.Save(msg);
```

and then this code:



```
    using (SmtpClient smtp = new SmtpClient())
    {
        smtp.DeliveryMethod = SmtpDeliveryMethod.Network;
        smtp.UseDefaultCredentials = false;
        smtp.EnableSsl = true;
        smtp.Host = "smtp.gmail.com";
        smtp.Port = 587;
        smtp.Credentials = new System.Net.NetworkCredential("sashak95@gmail.com", "1234567890");
        // send mail
        smtp.Send(message);
    }

    return RedirectToCurrentUmbracoPage();
}
```

basically takes the inputted values in the form and send them as an email to whatever email you have assigned it to: in our case it is sashak95@gmail.com.

Evaluation and Conclusion

To summarise up, we think that it was very useful project, because we got to learn a lot of new stuff that we can implement in future project we work on in Umbraco. We felt a little bit irritated by the many errors and problems we faced towards the end of the project, but somehow we managed to solve them in the end. The documentation of Umbraco being mostly in videos makes it hard to follow up at a specific topic that you are having problems with. As an overall opinion, we feel like this topic was useful for a future career in the field!