

Veggie Products

- en applikation til indtastning og søgning af vegetariske produkter



Skriftlig opgave i faget "Webudvikling - backend"

Skrevet af: Jane Jin Larsen

Uddannelse: Diplom i softwareudvikling

Underviser og vejleder: Kianoush Golipour

Afleveret: d. 02. juni 2021

Tegn uden mellemrum: 26513

Foto fra Webside: <https://i.imgur.com/ehj4khR.jpg>

Indholdsfortegnelse

Tema og problemstilling	2
Problemformulering	2
Udformning, Afgrænsning & værktøjer	2
Værktøjer	3
Emnebehandling	3
Mapning til database og model	4
Routing og mappe-struktur med Areas	5
Valideringer med data-annotations samt brug af komma	6
Authentication og authorization	7
Søge-funktion	10
Deployment til Azure Cloud	11
Konklusion	12
Perspektivering	13
Bilag	14
Litteraturliste, referencer og noter	32
BILAG 1: DIAGRAM OVER TABEL-RELATIONER	14
BILAG 2 : MVC-ARKITEKTUR	15
BILAG 3: LANDING PAGE - HOMEPAGE	16
BILAG 4: LISTE AF PRODUKTER	17
BILAG 5: 'LOG IN' MED EMAIL OG PASSWORD	18
BILAG 6: 'DETAILS' FOR PRODUKT	19
BILAG 7: TILFØJ NYT PRODUKT (LOGGET IND SOM ENDUSER)	20
BILAG 8: REDIGER PRODUKT (LOGGET IND SOM ENDUSER)	21
BILAG 9: LOG IN SOM ADMIN (MANAGERUSER)	22
BILAG 10: DROPDOWN- MENU MED KATEGORI OG BRUGERE	22
BILAG 11: LISTE MED KATEGORIER (LOGGET IND SOM MANAGERUSER)	23
BILAG 12: DETAILS (LOGGET IND SOM MANAGERUSER)	23
BILAG 13: REDIGER KATEGORI (LOGGET IND SOM MANAGERUSER)	24
BILAG 14: SLET KATEGORI (LOGGET IND SOM MANAGERUSER)	24
BILAG 15: LISTE AF BRUGERE (LOGGET IND SOM MANAGERUSER)	25
BILAG 16: TILFØJ NY BRUGER (LOGGET IND SOM MANAGERUSER)	25
BILAG 17: REGISTRER SIG SOM NY BRUGER	26
BILAG 18: MAPPE-STRUKTUR UNDER AREAS OG MODELS	27
BILAG 19: KLASSEN MED PAGINATEDLIST	28
BILAG 20: SETTINGS FOR APP SERVICE PLAN IN AZURE	29
BILAG 21: SETTINGS FOR AZURE SQL DATABASE	30
BILAG 22: PUBLISH TO AZURE	31

TEMA OG PROBLEMSTILLING

På grund af klimakrise, sundhedsinteresse eller dyreetik er de kødløse og plantebaseret måltider blevet en global trend, som også præger udvalget i supermarkeder. Der er kommet mange flere kødløse og plantebaseret produkter på markedet i de sidste par år. Det er dog stadig ikke alle supermarkeder, der fører de samme produkter, og udvalget kan være meget forskelligt. Så derfor kan det for forbrugeren nogle gange være lidt svært at orientere sig, og man kan også risikere at gå forgæves i et supermarked, fordi de ikke har produktet i deres sortiment. Derfor synes jeg, at der har manglet et system (fx en mobilapp eller hjemmeside), hvor man kan se, i hvilken butik man kan købe nogle forskellige plantebaseret produkter.

Af ovenstående grunde ser jeg derfor et behov for at lave en webapplikation for vegetariske produkter. Den skal fungere sådan, at alle slutbrugere kan taste ind, hvor man kan købe et bestemt produkt fx "Smørbar" i Kvickly. Eller "Alpro fløde" i Bilka - med adresse og pris. Det skal så være sådan, at man også kan søge efter et produkt, og så kan man finde ud af om de har det i den nærmeste dagligvarebutik.

PROBLEMFORMULERING

Et system der kan håndtere relevante data om vegetariske produkter samt at have søgefunktioner på produkt og kategori. Det skal derfor være muligt at gemme permanente data i en database samt vise data fra databasen.

Herunder er også følgende underemner.

- Administration af sidens brugere
- Funktioner med login og registrering
- Brug af Entity Framework i ASP.net Core

UDFORMNING, AFGRÆNSNING & VÆRKTØJER

Fokusområde er mest på webapplikationen med opbygningen af websider, som har funktioner, der gør det let for slutbrugeren at se en liste af produkter og søge produkter ud fra bestemte kriterier. Så der skal være de essentielle CRUD-operationer (create, read, update, delete), så brugere kan benytte funktioner som at oprette, indlæse, opdatere og slette et produkt. Der er desuden også funktioner med at administrere brugere af siden med forskellige roller(authorization) samt mulighed for at logge ind på siden (authentication).

Jeg har dog ikke fokuseret på forholdene omkring IT-sikkerheden i selve applikationen med fx kryptering eller personfølsomme data, idet jeg vil holde mit fokus på selve udviklingen af systemet

og implementering af kerne-funktionerne i web-applikationens back-end. På grund af opgavens omfang sigter jeg heller ikke på at opbygge en fuldt kørende enterprise-løsning i en kompleks arkitektur fra starten, og jeg tester heller ikke for performance, og laver derfor heller ikke fx unit-tests eller bruger-test.

Jeg kigger heller ikke så meget på design, stil og forståelighed i designet, men jeg tilsætter dog lidt i form af farver og billeder, og ellers er der kun basis css-styling, hvor det meste følger med i den auto-genereret kode-skabelon i bootstrap. Derfor opfylder web-applikationen heller ikke krav om fx webtilgængelighed.

VÆRKTØJER

Webapplikation laves med værktøjet *Visual Studio 2019* som et ASP.net Core MVC projekt med Entity Framework i .NET core 3.1.

Visual Studio-projektet uploades til *Github* (<https://github.com/webfav/VeggieProductsApp2.git>)

Projektet er desuden også publiceret på Azure Cloud, hvor jeg har en gratis basis-konto.

<https://veggieproductsapp2.azurewebsites.net/>

EMNEBEHANDLING

Jeg bruger skabelonen for ASP.net Core MVC Application sammen med Entity Framework, da det er ret hurtigt og fleksibelt at sætte op, og da det er indbygget i Visual Studio med meget autogenereret kode. Så der generes en model, som repræsenterer data-objekterne i databasen, og fordelene er bl.a., at man ikke behøver at skrive koden til data-forbindelserne hver gang. Entity Framework er et object-relational mapping (ORM) værktøj som er opdelt i 3 slags lag - en databasemodel (tabeller, visninger osv.), en konceptuel model, der repræsenterer objekterne i appen samt en mapping mellem disse to. Der gives en nem adgang til at oprette forbindelse til datakilder som fx SQL server ved en *connectionstring* som sættes i appsettings.json - klassen.

VeggieProductsApp2/appsettings.json

```
"ConnectionStrings": {  
  "DefaultConnection":  
    "Server=(localdb)\\mssqllocaldb;Database=VeggieProductsApp2;Trusted_Connection=True;MultipleActiveResultSets=true"  
},
```

Man sætter forbindelsen til *connectionstring* i Startup-klassen under ConfigureServices, hvor jeg gør brug af SQLserver og sætter en DbContext.

VeggieProductsApp2/Startup.cs

```
public void ConfigureServices(IServiceCollection services)  
{
```

```
services.AddDbContext<ApplicationDbContext>(options =>
    options.UseSqlServer(Configuration.GetConnectionString("DefaultConnection")));
```

MAPNING TIL DATABASE OG MODEL

Som der ses i bilag 1 har jeg valgt en opdeling i produkt og kategori, hvor properties oprettes i hver sin klasse under model. Derefter laves et DbSet under ApplicationDbContext, så jeg kan gøre brug af modelklasserne, når jeg henter context som dependency injection i controller-klassen.

Her er ApplicationDbContext med DbSet for Category, Product og ApplicationUser (markeret med gult). *ApplicationUser* forklares senere under Authentication (side 8).

VeggieProductsApp2/Data/ApplicationDbContext.cs

```
namespace VeggieProductsApp2.Data
{
    public class ApplicationDbContext : IdentityDbContext
    {
        public ApplicationDbContext(DbContextOptions<ApplicationDbContext> options)
            : base(options)
        {
        }
        public DbSet<Category> Category { get; set; }
        public DbSet<Product> Product { get; set; }
        public DbSet<ApplicationUser> ApplicationUser { get; set; }
    }
}
```

ApplicationDbContext hentes ind som dependency injection i controller-klassen.

VeggieProductsApp2/Areas/Admin/Controllers/ProductController.cs

```
private readonly ApplicationDbContext _context;
public CategoryController(ApplicationDbContext context)
{
    _context = context;
}
```

For at gøre brug af begge modeller *Product* og *Category* i et view, laver jeg også en viewmodel ProductVM. Da jeg skal bruge en liste af kategorier i et dropdown-felt, sætter jeg categories som IEnumerable.

VeggieProductsApp2/Models/ViewModels/ProductVM.cs

```
namespace VeggieProductsApp2.Models.ViewModels
{
    public class ProductVM
    {
        public Product Product { get; set; }
        public IEnumerable<Category> Categories { get; set; }
    }
}
```

For at vise properties fra begge modeller gennem viewmodel kan jeg herefter gøre brug af eager-loading med .Include for at inkludere Category til Product. Her fra fx Details-actionmetoden.

VeggieProductsApp2/Areas/Admin/ Controllers/ProductController.cs

```
ProductVM.Product = await _context.Product.Include(m => m.Category)
.SingleOrDefaultAsync(m => m.Id == id);
```

For at binde til ProductVM og bruge den i de andre action-metoder, kan jeg sætte det i fx Product-controller-klassen med [BindProperty], og herefter sætte den i constructoren.

VeggieProductsApp2/Areas/Admin/Controllers/ProductController.cs

```
[BindProperty]
public ProductVM ProductVM { get; set; }

public ProductController(ApplicationDbContext context)
{
    _context = context;

    ProductVM = new ProductVM()
    {
        Categories = _context.Category,
        Product = new Models.Product()
    };
}
```

ROUTING OG MAPPE-STRUKTUR MED AREAS

Jeg har lavet views og controllers i Area mappe-struktur for at lave en mere logisk organisering, hvor det så kun er models, der bliver delt af de forskellige uafhængige grupper i areas. (jvf. evt. bilag 2). *Identity* bliver scaffoldet med autogenereret kode under Area som razor-pages. Jeg laver så to mere grupper i Areas med hhv. *Admin* og *Homepage*. Under *Admin*-mappen lægger jeg de 3 hoved-controllers CategoryController, ProductController og UserController med de dertilhørende views (se bilag 18).

Jeg har tilføjet area med *Homepage* i routing. Dette sættes i UseEndpoints metoden i startUp-klassen (markeret med gult).

	<pre>app.UseEndpoints(endpoints => { endpoints.MapControllerRoute(name: "default", pattern: "{area=Homepage}/{controller=Home}/{action=Index}/{id?}"); endpoints.MapRazorPages(); });</pre>
--	---

Hver controller-klasse skal så have tilføjet attributten fx `[Area("Homepage")]` eller `[Area("Admin")]` i toppen af klassen for at sætte, hvilket *Area* klassen tilhører.

VALIDERINGER MED DATA-ANNOTATIONS SAMT BRUG AF KOMMA

En fordel ved ASP.net Core er også, at man kan gøre brug af data-annotations, som sikrer validation server-side, og derved også er med til at sikre sig mod forkert data-input i applikationens frontend, og det er samtidig også en hjælp til slutbrugeren. Så man kan fx sikre et korrekt match med den angivne datatype, så fx Dato er af typen `[Date]` osv.

Jeg har fx sat attributten `[Required]` sammen med en fejlmeddelelse på `ProductName` `[Required(ErrorMessage = "Produktets navn skal tilføjes!")]`. Så hvis der ikke blev indtastet produkt i tekstboksen, ville denne fejlmeddelelse med `"Produktets navn skal tilføjes!"` poppe op. Fejlmeddelelsen bliver bundet op i ``, hvilket er på klient-siden i view.

Da der også er forskel på, hvordan man skriver valuta i mange europæiske lande og i USA, har jeg også sat et script ind, der accepterer brug af komma som separator i input-feltet. Dette script overskriver en del af den eksisterende jquery validation, og det er derfor sat i både Create-view og Edit-view. USA skriver pris med punktum fx 10.00 \$, mens vi i DK skriver det med komma 10,00 kr. Jeg sætter så `id="currencyTextBox"` i taghelper-koden for Pris (jvf. evt. bilag 8).

AUTHENTICATION OG AUTHORIZATION

Her ses en oversigt over de forskellige sider i applikationen, hvor man starter med indgangs-siden Homepage (se også bilag 1).

<i>Website - sideoversigt</i>	
PAGE	ITEMS
Homepage /Index (se bilag 3)	<ol style="list-style-type: none"> 1. Velkommen til siden - kort beskrivelse 2. Navigation bar <ol style="list-style-type: none"> a. Management (kun adgang for manager) b. List c. Login
Liste af produkter - Page /Admin/Product /Admin/Product/Details/8 (se bilag 4)	<ol style="list-style-type: none"> 1. Product list <ol style="list-style-type: none"> a. Tilføj ny (kun adgang for user) b. Rediger (kun adgang for user) c. Details d. Slet (kun adgang for manager) 2. Søge-funktion
Kategori - Page / Admin/Category / Admin/Category/Edit/1 (kun adgang for manager) (se bilag 11)	<ol style="list-style-type: none"> 1. Kategori- list <ol style="list-style-type: none"> a. Tilføj ny b. Rediger c. Details d. Slet
Brugere - Page / Admin/User (kun adgang for manager) (se bilag 15, 16)	<ol style="list-style-type: none"> 1. Liste af brugere 2. Lås /lås op 3. Tilføj ny (redirect til Register page)
Login - Page /Identity/Account/Login (se bilag 9)	<ol style="list-style-type: none"> 1. Login
Register - Page /Identity/Account/Register (se bilag 17)	<ol style="list-style-type: none"> 2. Register

Som det ses er der CRUD-operationer samt søge-funktion på produktliste-siden. Men denne side har også en opdeling af hvilken rolle, der har adgang. Så det er manager/admin, der har adgang til det hele, mens en registreret bruger (Enduser) kun har adgang til at kunne tilføje et nyt produkt eller redigere et eksisterende produkt. En anonym bruger har kun adgang til at kunne se selve listen og detaljer, og kan derved ikke gøre meget andet på siden. Hvis en anonym bruger trykker for at kunne oprette et nyt produkt, så skal de registrere sig og bliver derved omdirigeret til registreringssiden. Manager eller admin har derimod adgang til at kunne tilføje nye brugere samt lave operationer på kategorier med CRUD. Denne opdeling i adgang laves med authorization og authentication, hvor man gør brug af klasserne i *Identity*. I oprettelsen af MVC-projektet kan tilvælges "Individual User

Accounts" som automatisk opsætter *Identity* i projektet. Herefter kan alle *Identity*-tabellerne genereres, når der første gang pushes migration til DB via Entity Framework. Så pakken med Entity Framework er allerede med i projektet, og man skal ikke selv installere den "nuget package". Pakke-referencerne ses også i .csproj-filen samt i mappen Dependencies.

For at gøre brug af *Identity* sættes den som service i startup-klassen med både *IdentityUser* og *IdentityRole*.

VeggieProductsApp2/Startup.cs

```
services.AddIdentity<IdentityUser, IdentityRole>()
    .AddDefaultTokenProviders()
    .AddEntityFrameworkStores<ApplicationDbContext>();
```

Jeg gør brug af de autogenereret tabeller til *AspNetUsers* og *AspNetUserRoles* samt til login.

Klasser og tabeller ses i bilag 1.

Jeg har dog lavet en tilpasning til *AspNetUsers* for også at have *Zipcode* og *Fullname* med. Derfor har jeg oprettet en klasse med *ApplicationUser* som en udvidelse fra *IdentityUser*, og derefter har jeg modificeret Register-siden med de nye properties.

VeggieProductsApp2/Areas/Identity/Pages/Account/Register.cshtml.cs

```
[Required]
[Display(Name = "Navn")]
public string FullName { get; set; }
```

```
[Required]
[Display(Name = "Postnr")]
public string Zipcode { get; set; }
```

```
//changed from IdentityUser to ApplicationUser
var user = new ApplicationUser
{
    UserName = Input.Email,
    Email = Input.Email,
    FullName = Input.FullName,
    Zipcode = Input.Zipcode
};
```

Derefter har jeg sat de 2 roller i en klasse som konstanter i en *Users*-klasse.

VeggieProductsApp2/ManagedUsers/Users.cs

```
namespace VeggieProductsApp2.ManagedUsers
{
    public static class Users
    {
        public const string ManagerUser = "Manager";
        public const string EndUser = "EndUser";
    }
}
```

Disse 2 roller kan jeg herefter sætte som attributter før en action-metode for at angive hvortil den pågældende rolle har adgang. Hvis det er en anonym bruger, dvs. at alle har adgang, kan man sætte attributten `[AllowAnonymous]`.

I fx *Create*-klassen er der behov for at både Manager og Enduser har adgang, og derfor er der brug for en attribut, der samler disse 2 roller. De er derfor samlet i attributten

[AuthorizeAdminOrEndUser] ved hjælp af denne klasse som arver fra *AuthorizeAttribute*.

VeggieProductsApp2/Areas/Admin/Controllers/ProductController.cs

```
public class AuthorizeAdminOrEndUser : AuthorizeAttribute
{
    public AuthorizeAdminOrEndUser()
    {
        //taking both roles to authorize
        Roles = Users.ManagerUser + "," + Users.EndUser;
    }
}
```

Disse roller gør jeg også brug af i navigations-menuen, så det kun er ManagerUser, der har adgang til dropdown-menuen *Produkt Management* med Kategori og Brugere (se også bilag 10). For at få vist denne dropdown-menu og dermed få adgang skal man være logget ind som Admin, der har rollen som ManagerUser.

VeggieProductsApp2/Views/Shared/_Layout.cshtml

```
@if (User.IsInRole(Users.ManagerUser))
{
    <li class="nav-item dropdown text-white">
        <a class="nav-link dropdown-toggle" href="#" id="navbarDropDownMenuLink" role="button"
            data-toggle="dropdown" aria-haspopup="true" aria-expanded="false">
            Produkt Management</a>
        <div class="dropdown-menu" aria-labelledby="navbarDropDownMenuLink">
            <a class="dropdown-item" asp-action="Index" asp-controller="Category"
                asp-area="Admin">Kategori</a>
            <a class="dropdown-item" asp-area="Admin" asp-controller="User"
                asp-action="Index">Brugere</a>
        </div>
    </li>
}
```

Det er også kun manager, der har mulighed for at låse en brugers adgang op, hvis denne ved et uheld er kommet til at skrive forkert password for mange gange. Og manager kan også låse en brugers adgang, hvis der fx er behov for at brugeren skal udelukkes fra indtastning og redigering af data. Dette gøres ved hjælp af *LockoutEnd*.

VeggieProductsApp2/Areas/Admin/Views/User/Index.cshtml

```
<td>
    @*user is not locked*@
    @if (item.LockoutEnd == null || item.LockoutEnd < DateTime.Now)
    {
        <a class="btn btn-success text-white" asp-action="LockingUser"
            asp-route-id="@item.Id">Lås bruger</a>
    }
    else
    //User is locked
    {
        <a class="btn btn-danger text-white" asp-action="UnLockingUser"
            asp-route-id="@item.Id">Oplås bruger</a>
    }
</td>
```

SØGE-FUNKTION

Ved opbygning af funktionerne med søgestreng og grupperet sider har jeg hovedsageligt fulgt denne tutorial "Add sorting, filtering, and paging - ASP.NET MVC with EF Core"

<https://docs.microsoft.com/en-us/aspnet/core/data/ef-mvc/sort-filter-page?view=aspnetcore-5.0>

Der tilføjes en søgestreng som parameter "searchString" til index-metoden. Søgestrengens værdi, kommer fra et input tekstfelt, som tilføjes til index-view. Til selve LINQ-sætningen tilføjes en where-clause, der kun vælger produkter, hvis *ProductName* eller *CategoryName* indeholder "searchString". Erklæringen udføres kun, hvis der er en værdi at søge efter dvs. ikke null eller tom.

VeggieProductsApp2/Areas/Admin/Controllers/ProductController.cs

```
// .....  
  
var productsContext = from s in _context.Product.Include(p => p.Category)  
                        select s;  
if (!String.IsNullOrEmpty(searchString))  
{  
    productsContext = productsContext.Where(s =>  
        s.ProductName.Contains(searchString)  
        || s.Category.CategoryName.Contains(searchString));  
}  
  
// ....  
  
int pageSize = 3;  
return View(await PaginatedList<Product>.CreateAsync(productsContext  
    .Include(p => p.Category).AsNoTracking(), pageNumber ?? 1, pageSize));
```

I slutningen af index-metoden bliver forespørgslen til *PaginatedList.CreateAsync*-metoden konverteret til en enkelt side af products i en collection-type. Her har jeg sat *pageSize* til 3, så der er kun 3 records på en enkelt side. Denne side med product sendes derefter til view.

Her er Søgefelt og knap i Index-view på produkt.

VeggieProductsApp2/Areas/Admin/Views/Product/Index.cshtml

```
<p>  
    <input class="form-control mr-sm-2" type="text" name="SearchString"  
        placeholder="Indtast produkt eller kategori" value="@ViewData["CurrentFilter"]" />  
    <button class="btn btn-primary my-2 my-sm-0" type="submit">Søg</button>  
    <a class="btn btn-outline-primary" asp-action="Index">Tilbage til fuld liste</a>  
</p>
```

Bemærk også, at selve URL'en indeholder søgestrengen, når man søger efter produkt eller kategori fx her efter søgning på fx "Vegan".

<https://localhost:44357/Admin/Product?SearchString=Vegan>

For at opsætte de grupperede sider, hvor man kan gå frem og tilbage i listen, oprettes der en `PaginatedList`-klasse (se bilag 19).

Knapperne i view vises med tag-helpers, hvor man i `pageNumber` går frem med +1 eller tilbage med -1.

VeggieProductsApp2/Areas/Admin/Views/Product/Index.cshtml

```
@{ var prevDisabled = !Model.HasPreviousPage ? "disabled" : "";
    var nextDisabled = !Model.HasNextPage ? "disabled" : ""; }

<a asp-action="Index"
    asp-route-sortOrder="@ViewData["CurrentSort"]"
    asp-route-pageNumber="@((Model.PageIndex - 1))"
    asp-route-currentFilter="@ViewData["CurrentFilter"]"
    class="btn btn-primary @prevDisabled">
    Forrige side
</a>
<a asp-action="Index"
    asp-route-sortOrder="@ViewData["CurrentSort"]"
    asp-route-pageNumber="@((Model.PageIndex + 1))"
    asp-route-currentFilter="@ViewData["CurrentFilter"]"
    class="btn btn-primary @nextDisabled">
    Næste side
</a>
```

DEPLOYMENT TIL AZURE CLOUD

Jeg har valgt at cloud-hoste og deploye til Azure-cloud, og til dette projekt er det foreløbig tilstrækkeligt med en sql-server til databasen og en app service til at hoste websiderne.

For at kunne tilgå websiden med admin-login første gang er denne admin-bruger *initialized* i en anden klasse for at *seed* databasen med admin-data. Admin-brugeren bliver derfor oprettet med username som "Admin@gmail.com" og password "Admin!0".

VeggieProductsApp2/Data/DbInitializer.cs

```
//default user Admin gets added to db
_userManager.CreateAsync(new ApplicationUser
{
    UserName = "admin@gmail.com",
    Email = "admin@gmail.com",
    FullName = "Admin",
    Zipcode = "1000",
    EmailConfirmed = true
}, "Admin!0").GetAwaiter().GetResult();
```

Herefter hentes *DbInitializer* ind i Startup-klassen som en service. Og denne bliver så kaldt i configure-metoden, så man under runtime kan logge ind med admin-login.

```
VeggieProductsApp2/Startup.cs

services.AddScoped<IDbInitializer, DbInitializer>();
//....

//invoke method to seed db
dbInitializer.Initialize();
```

Det er meget nemt at udgive sin webapplikation på Azure ved hjælp af Visual Studio, hvis man allerede har oprettet en konto. I processen med *publish*, er det nødvendigt at definere en App service plan med en ressourcegroup og den rette region i hosting plan (se bilag 20). Da det også er nødvendigt med en database er en SQL server også defineret i samme ressourcegroup (se bilag 21). Hvis der ikke sker nogle fejl-konfigurationer, så bliver der oprettet en azurewebsite <https://veggieproductsapp2.azurewebsites.net/>

KONKLUSION

Sammenfattende kan det konkluderes, at overordnet set fungerer webapplikationen efter hensigten med opbygning i entity framework i Asp.net Core.

Der er funktionalitet for følgende:

- CRUD-operationer på både produkt og kategori.
- Det er også lykkedes med at lave søgefunktioner på produkt og kategori, som kan ses i en liste, hvor der tilmed er visning i grupperet sider.
- Der er også funktioner med login og registrering for brugere.
- En funktion med administration af sidens brugere, hvortil der også er implementeret adgangskontrol.

Efter deployment til Azure viser der sig dog stadig at være lidt problemer med at gemme og vise pris på den rette måde med den rigtige cultureinfo. Som et eksempel kan man se at en pris-indtastning af fx 10,50 kr bliver gemt som 1050 og vist i listen som 1,050.00. Så den rigtige globalization setting med "da-DK" burde også blive implementeret. Et projekt er selvfølgelig også en fortløbende proces med *Continuous Integration* (CI) og *Continuous Deployment/delivery* (CD), som jeg dog ikke har gået ind i her.

Der viser sig også en fejl ved visning af produktnavn i listens header, da denne mangler. Så der kunne sagtens gøres mere ud af den sekundære funktionalitet samt styling og design.

Overordnet set har webapplikationen en fungerende funktionalitet, men for at sætte den i fuld produktion er der stadig noget udviklingsarbejde og optimering af løsningen, idet jeg ikke har lavet test, logging og større sikkerhedsmæssige foranstaltninger af siden som forsvar mod hacker-angreb.

PERSPEKTIVERING

Gennem dette projekt har mit udgangspunkt som sagt været at begrænse mig til udviklingen af webapplikationens CRUD-operationer samt søge-funktion. Hvis tiden og kapaciteten tillod det, kunne der laves en del forbedringsarbejde.

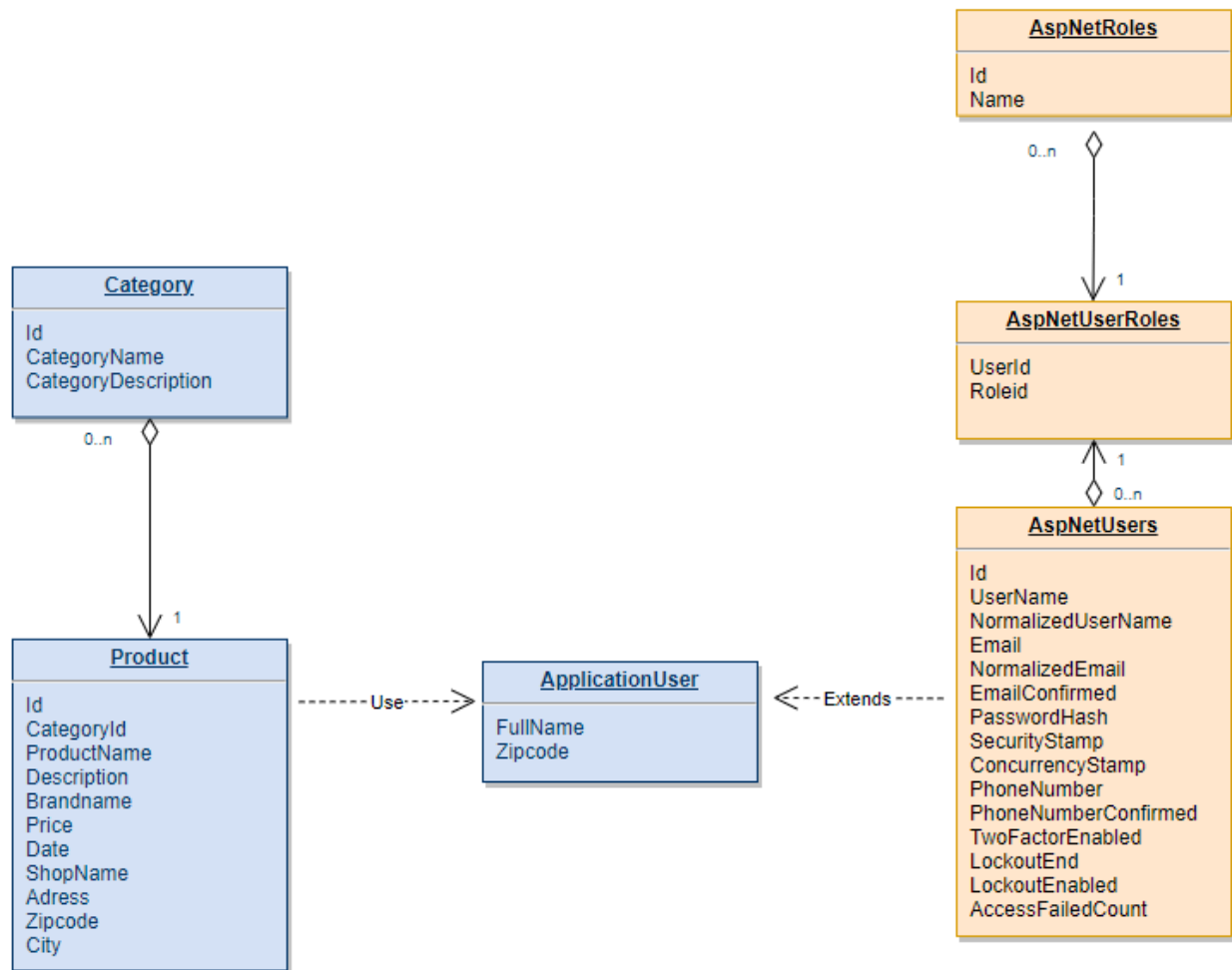
Man kunne lave et bedre model-design med adskillelse af shop og produkt, så disse er uafhængige af hinanden. For så kan man gå ud fra at én butik kan selvfølgelig have mange forskellige slags produkter i sit sortiment, og omvendt kan ét produkt findes i mange forskellige slags butikker. Et bedre model-design ville også give mulighed for at søge mere uafhængigt på en bestemt butik på fx postnr. Eller man kunne lave funktioner med en søgning i et kort med GPS-positioner.

En funktion med upload af fotos ville også kunne forbedre brugen af app betydeligt, da det så er langt lettere at se, hvordan produktet ser ud. Måske kunne en forbruger tage et billede med sin mobil som så uploades via appen. Her kan man også med fordel gøre brug af fx blobstorage i Azure Cloud til at gemme billed-filerne.

Jeg kunne også gøre brug af den indbyggede to-faktor authentication(2FA) eller Multi-factor authentication (MFA), som er en proces, hvor en bruger kan blive anmodet om yderligere former for identifikation under login. Dette kan fx være at indtaste en engangs-kode fra en mobiltelefon eller give en fingeraftryksscanning. Når der er lavet en anden form for godkendelse, vil sikkerheden forbedres, og en hacker kan ikke så let angribe.

Endelig kan man sige, at der er ikke tale om en ny type it-produkt, men selve indholdet er måske stadig lidt niche-præget med vegetariske produkter, og her kan IT også være med til at innovere omkring markedet i selve bæredygtigheds-tankegangen og den grønne bølge.

Bilag 1: Diagram over tabel-relationer



Bilag 2 : MVC-arkitektur

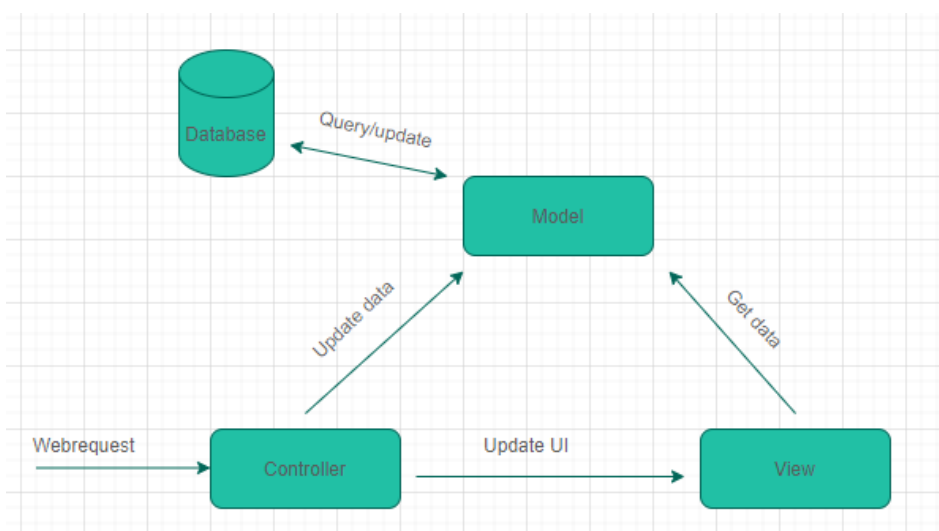
Kort fortalt er MVC arkitektur en logisk opbygning i model-view-controller, hvor man adskiller model, kontrol og view i hvert sit ansvarsområde. Denne opbygning i løs kobling gør det nemmere at lave modifikationer i en del af koden, hvor man fx kan modificere kun view-delen uden at ændre i controller eller model. MVC gør det også nemmere at genbruge kode fx kan der laves flere views til den samme model. Samtidig er alle views grupperet sammen, da disse moduler funktionsmæssigt er hører sammen, og ligeledes er alle controller handlinger grupperet sammen. MVC er også med til at inddrage design-princippet om "separation of concerns" i adskillelsen af frontend og backend, og der er ikke direkte adgang til databasen.

Model: her håndteres data og de krav, der er til de enkelte data i klasser samt lagring og afhentning fra databaser til et program. Her ligger klassen for Product og klassen for Category, hvilket svarer til de 2 tabeller i databasen.

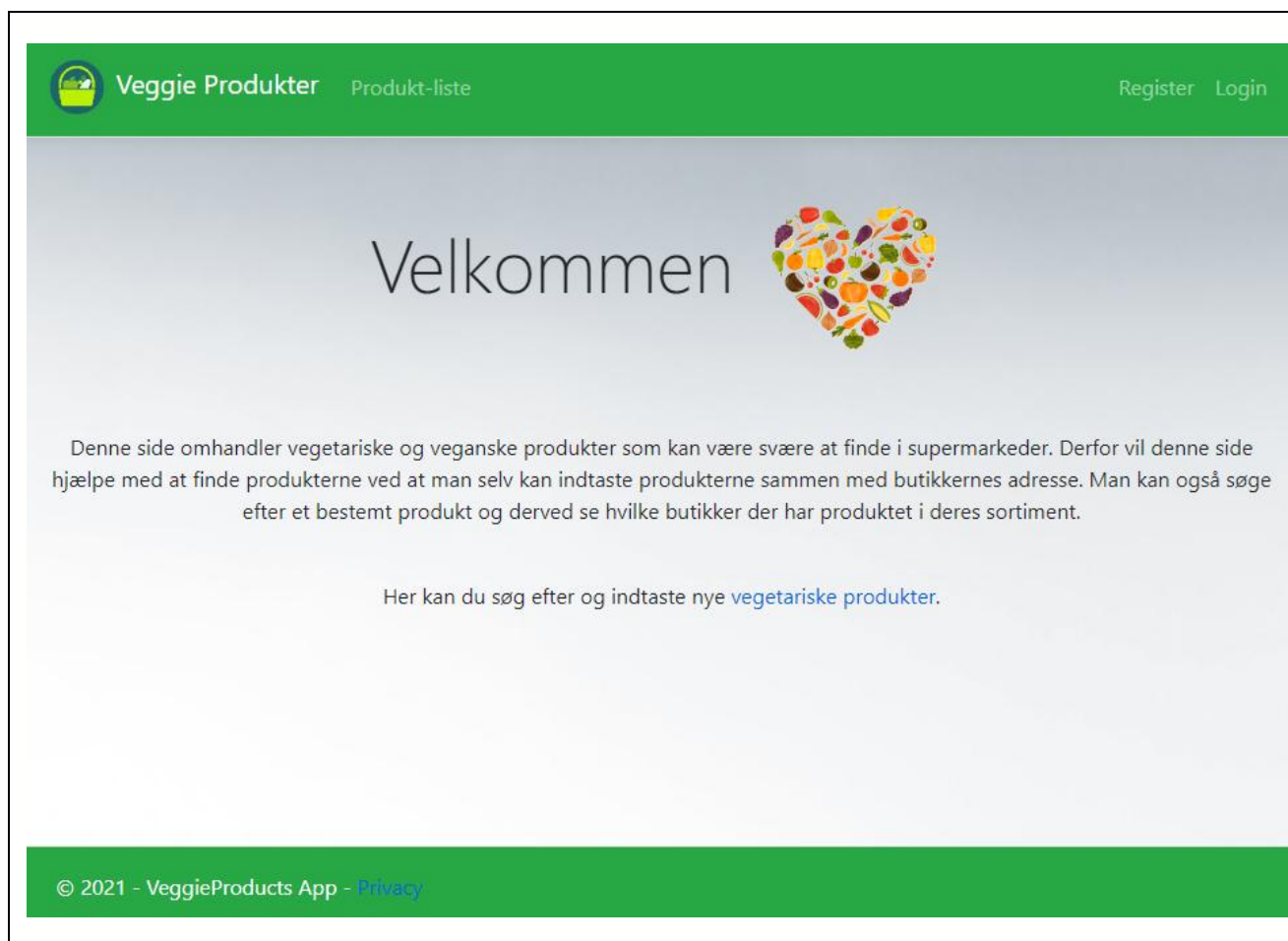
View: her håndteres, hvad brugeren skal se på siden. Typisk er view lavet ud fra modellens properties. og data. I applikationen er der oprettet flere views til bla. CRUD-operationer, men også til login, fejl og management af brugere og kategori.

Controller: her håndteres forespørgsler og svar fra brugeren eller fra andre sider. Controller fungerer som mellemed mellem model og view. Her laves fx action-metoderne til siderne for index, create, edit, details og delete.

Figuren herunder viser MVC flow med routing, hvor controller-delen modtager forespørgsel og svar fra brugeren, så det er controlleren, der videresender data eller handlinger (actions) til view-siderne, som brugeren ser og interagerer med. View kan så hente data fra modellen eller udføre controllerens action.




Bilag 3: Landing page - Homepage



Bilag 4: Liste af produkter

Liste af produkter med søge-funktion og mulighed for at se detaljer. Tilføj ny og rediger er forbeholdt brugere (EndUser) som har registreret sig og skal logge ind på siden.

 **Veggie Produkter** Produkt-liste Register Login


Tryk her for at tilføje et nyt produkt til listen

Søg Tilbage til fuld liste

	Beskrivelse	Mærke	Pris	Dato	Butik	Adresse	Postnr	By	Kategori	
Beyond Meatballs	Plantebaseret kødboller, italiensk smag	Beyond Meat	10,00	06-05-2021	Bilka	Niels Bohrs Alle 150	5230	Odense M	Vegan	<div>Rediger Detaljer</div>
Havredrik	Økologisk plantebaseret havredrik	Silk	8,00	28-05-2021	Netto	Hjørringvej 144	9900	Frederikshavn	Dairy-free	<div>Rediger Detaljer</div>
Meatless Chicken Tenders	Kødløse stykker til at koge eller stege	Quorn	25,00	07-05-2021	Fakta	Jægersborg Alle 14-16	2920	Charlottenlund	Vegan	<div>Rediger Detaljer</div>

Forrige side Næste side

Bilag 5: 'Log in' med email og password

 Veggie Produkter [Produkt-liste](#) [Register](#) [Login](#)

Log in

Use a local account to log in.

Email

Password

☐ Husk mig?

[Log in](#)

[Har du glemt dit password?](#)


[Register som ny bruger](#)

Use another service to log in.

There are no external authentication services configured. See [this article](#) for details on setting up this ASP.NET application to support logging in via external services.

© 2021 - VeggieProducts App - [Privacy](#)

Bilag 6: 'Details' for produkt

 **Veggie Produkter** [Produkt-liste](#) [Register](#) [Login](#)


Her kan du se detaljer om produktet

Produkt

Produkt	Beyond Meatballs
Beskrivelse	Plantebaseret kødboller, italiensk smag
Mærke	Beyond Meat
Pris	10,00
Dato	06-05-2021
Butik	Bilka
Adresse	Niels Bohrs Alle 150
Postnr	5230
By	Odense M
Kategori	Vegan
Beskrivelse	Produkter uden kød og mejeri-indhold

[Rediger](#) | [Gå tilbage til listen](#)

Bilag 7: Tilføj nyt produkt (logget ind som EndUser)

 **Veggie Produkter** Produkt-liste Hello bruger2@gmail.com! Logout

Indtast nyt vegetarisk produkt

Produkt

Beskrivelse

Mærke

Pris

Dato

Butik

Adresse

Postnr

By


Kategori

-- Vælg Kategori --

Gem

Back to List

Bilag 8: Rediger produkt (logget ind som EndUser)

 **Veggie Produkter** Produkt-liste Hello bruger2@gmail.com! Logout

Rediger produkt

Produkt

Beskrivelse

Mærke

Pris

Dato

Butik

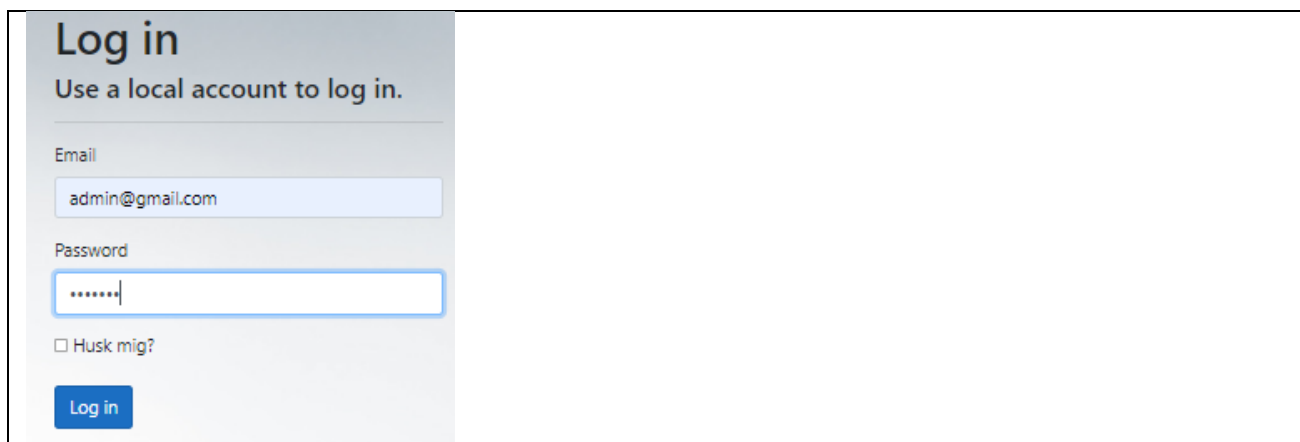
Adresse

Postnr

By

Kategori

Bilag 9: Log in som Admin (ManagerUser)



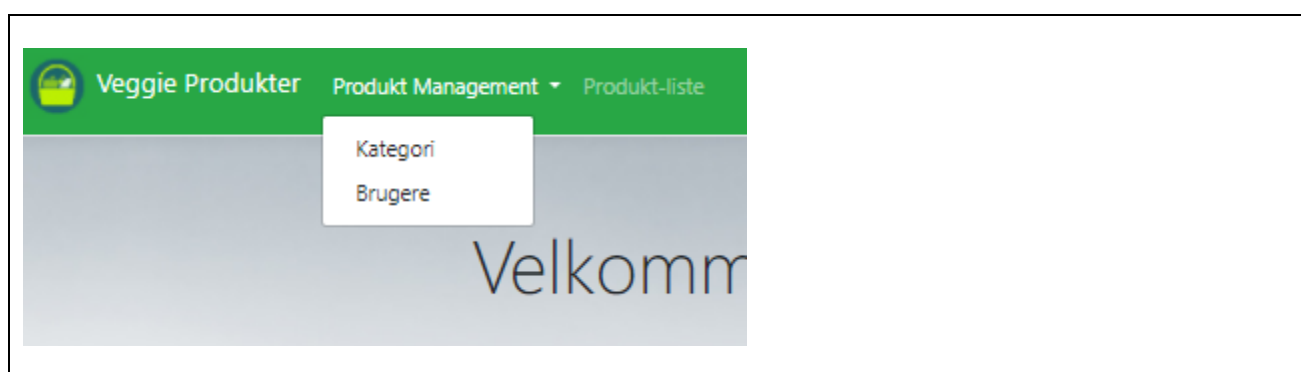
Log in
Use a local account to log in.

Email

Password

☐ Husk mig?

Bilag 10: Dropdown- menu med kategori og brugere



Bilag 11: Liste med kategorier (logget ind som ManagerUser)

Kategori-Liste			Tilføj ny
Kategori	Beskrivelse		
Dairy-free	Produkter uden mejeri-indhold	Rediger	Detaljer Slet
Vegan	Produkter uden kød og mejeri-indhold	Rediger	Detaljer Slet
Glutenfri	Produkter uden gluten	Rediger	Detaljer Slet
Eggfree	Produkter uden æg	Rediger	Detaljer Slet
Pesche	Produkter med fisk	Rediger	Detaljer Slet
Nuts-free	Produkter uden nødder	Rediger	Detaljer Slet

Bilag 12: Details (logget ind som ManagerUser)

Details	
Kategori	
Kategori	Dairy-free
Beskrivelse	Produkter uden mejeri-indhold
Rediger Tilbage til listen	

Bilag 13: Rediger kategori (logget ind som ManagerUser)

Rediger kategori

Kategori

Dairy-free

Beskrivelse

Produkter uden mejeri-indhold

Gem

Tilbage til listen

Bilag 14: Slet kategori (logget ind som ManagerUser)

Er du sikker på at du vil slette ?

Kategori

Kategori

Dairy-free

Beskrivelse

Produkter uden mejeri-indhold

Slet | Tilbage til listen

Bilag 15: Liste af brugere (logget ind som ManagerUser)

Veggie Produkter

Produkt Management

Produkt-liste

Hello admin@gmail.com! Logout

Brugerliste

Tilføj ny bruger

Navn	Postnr	Email	
Testbruger	8000	testbruger@gmail.com	Oplås bruger
SlutBruger	5000	bruger@gmail.com	Lås bruger
Bruger2	2300	bruger2@gmail.com	Lås bruger

Bilag 16: Tilføj ny bruger (logget ind som ManagerUser)

Opret ny bruger

Navn

Testbruger

Postnr

5000

Email

testbruger@gmail.com

Password

.....

Confirm password

.....

☒ EndUser ☐ Manager

Register ny bruger

Bilag 17: Registrer sig som ny bruger

Opret ny bruger

Navn	<input type="text"/>
Postnr	<input type="text"/>
Email	<input type="text"/>
Password	<input type="password"/>
Confirm password	<input type="password"/>

Register ny bruger

Bilag 18: Mappe-struktur under Areas og models

<ul style="list-style-type: none">Areas<ul style="list-style-type: none">Admin<ul style="list-style-type: none">Controllers<ul style="list-style-type: none">CategoryController.csProductController.csUserController.csViews<ul style="list-style-type: none">Category<ul style="list-style-type: none">Create.cshtmlDelete.cshtmlDetails.cshtmlEdit.cshtmlIndex.cshtmlProduct<ul style="list-style-type: none">Create.cshtmlDelete.cshtmlDetails.cshtmlEdit.cshtmlIndex.cshtmlUser<ul style="list-style-type: none">Index.cshtml_ViewImports.cshtml_ViewStart.cshtmlHomepage<ul style="list-style-type: none">Controllers<ul style="list-style-type: none">HomeController.csViews<ul style="list-style-type: none">Home<ul style="list-style-type: none">Index.cshtmlPrivacy.cshtml_ViewImports.cshtml_ViewStart.cshtmlIdentity<ul style="list-style-type: none">Pages<ul style="list-style-type: none">Account<ul style="list-style-type: none">Manage	<ul style="list-style-type: none">Models<ul style="list-style-type: none">ViewModels<ul style="list-style-type: none">ProductVM.csApplicationUser.csCategory.csErrorViewModel.csProduct.csViews<ul style="list-style-type: none">Shared<ul style="list-style-type: none">_Layout.cshtml_LoginPartial.cshtml_ValidationScriptsPartial.cshtmlError.cshtml_ViewImports.cshtml_ViewStart.cshtml
---	---

Bilag 19: Klassen med PaginatedList

```
namespace VeggieProductsApp2
{
    public class PaginatedList<T> : List<T>
    {
        public int PageIndex { get; private set; }
        public int TotalPages { get; private set; }

        public PaginatedList(List<T> items, int count, int pageIndex, int pageSize)
        {
            PageIndex = pageIndex;
            TotalPages = (int)Math.Ceiling(count / (double)pageSize);

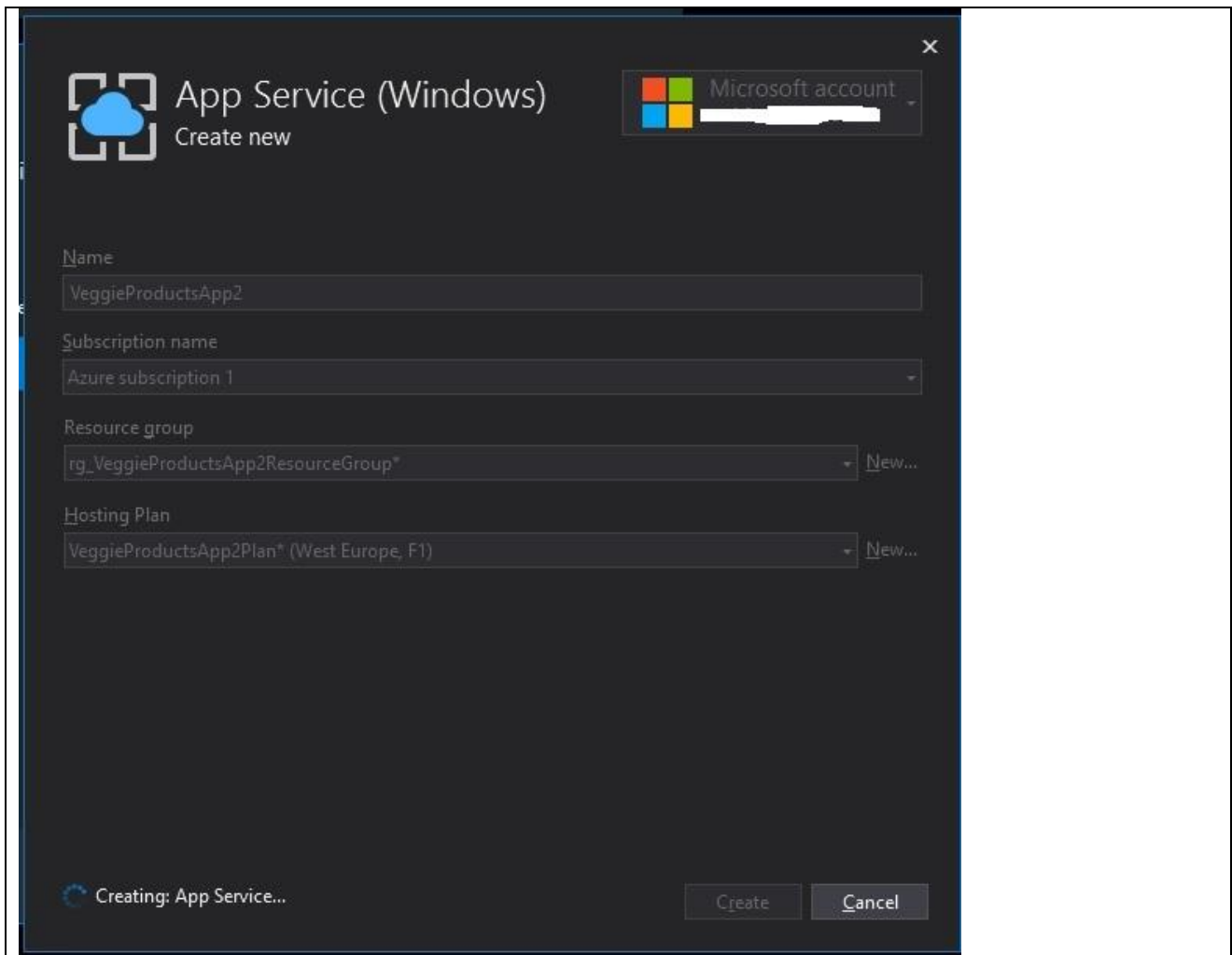
            this.AddRange(items);
        }

        public bool HasPreviousPage
        {
            get
            {
                return (PageIndex > 1);
            }
        }

        public bool HasNextPage
        {
            get
            {
                return (PageIndex < TotalPages);
            }
        }

        public static async Task<PaginatedList<T>> CreateAsync(IQueryable<T> source, int
pageIndex, int pageSize)
        {
            var count = await source.CountAsync();
            var items = await source.Skip((pageIndex - 1) *
pageSize).Take(pageSize).ToListAsync();
            return new PaginatedList<T>(items, count, pageIndex, pageSize);
        }
    }
}
```

Bilag 20: Settings for App service plan in Azure



The screenshot shows the 'App Service (Windows)' 'Create new' dialog box. At the top left is the App Service icon (a blue cloud with four white squares around it) and the text 'App Service (Windows)' with a 'Create new' link below it. At the top right is a 'Microsoft account' dropdown menu. The main area contains four fields: 'Name' with the value 'VeggieProductsApp2', 'Subscription name' with the value 'Azure subscription 1', 'Resource group' with the value 'rg_VeggieProductsApp2ResourceGroup*' and a 'New...' link, and 'Hosting Plan' with the value 'VeggieProductsApp2Plan* (West Europe, F1)' and a 'New...' link. At the bottom left is a progress indicator 'Creating: App Service...' with a circular arrow icon. At the bottom right are 'Create' and 'Cancel' buttons.

App Service (Windows)
Create new

Microsoft account

Name
VeggieProductsApp2

Subscription name
Azure subscription 1


Resource group
rg_VeggieProductsApp2ResourceGroup* New...

Hosting Plan
VeggieProductsApp2Plan* (West Europe, F1) New...

Creating: App Service...

Create Cancel

Bilag 21: Settings for Azure SQL database



Azure SQL Database

Create new

Microsoft account
[redacted]

Database name

VeggieProductsApp2_db

Subscription name

Azure subscription 1

Resource group

rg_VeggieProductsApp2ResourceGroup (West Europe)

New...

Database server

veggieproductsapp2dbserver* (West Europe)

New...

Database administrator username (must have permissions to create)

adminuser

Database administrator password


••••••••

Export...

Create


Cancel

Bilag 22: Publish to Azure

 VeggieProductsApp2 - Web Deploy.pubxml
Azure App Service (Windows)


Publish

+ New More actions ▾


 Ready to publish.

Settings


Configuration

Release 


Target Framework

netcoreapp3.1 

Deployment Mode

Framework-Dependent 


Target Runtime

Portable 

Show all

Hosting

Subscription


0e44e6b9-dfec-44c1-98c1-71ad326c9f87 

Resource group


rg_VeggieProductsApp2ResourceGroup

Resource name


VeggieProductsApp2

Site: <https://veggieproductsapp2.azurewebsites.net> 

Service Dependencies

 Azure SQL Database: VeggieProductsApp2_db

Connection string name: DefaultConnection

 Configured

LITTERATURLISTE, REFERENCER OG NOTER

Mit repository ligger på Github, med Visual Studio projektet og den skriftlige opgave

<https://github.com/webfav/VeggieProductsApp2.git>

Projektet er published på Azure Cloud. <https://veggieproductsapp2.azurewebsites.net/>

Tools/Værktøjer:

- **Flowchart Maker & Online Diagram Software**
URL: <https://app.diagrams.net/>
Sidst set 30-05-2021
- **Visual Studio Community 2019**
Downloades fra <https://visualstudio.microsoft.com/vs/>

Websider:

- **Entity Framework**
https://en.wikipedia.org/wiki/Entity_Framework
Sidst set 11/04-21
- **Create, read, update and delete**
URL: https://en.wikipedia.org/wiki/Create,_read,_update_and_delete
Sidst set 30-05-2021
- **What is the MVC pattern?**
URL: <https://docs.microsoft.com/en-us/aspnet/core/mvc/overview?view=aspnetcore-5.0>
Sidst set 30-05-2021
- **Areas in ASP.NET Core**
URL: <https://docs.microsoft.com/en-us/aspnet/core/mvc/controllers/areas?view=aspnetcore-5.0>
Sidst set 31-05-2021
- **Eager loading of Related Data**
URL: <https://docs.microsoft.com/en-us/ef/core/querying/related-data/eager>
Sidst set 31-05-2021

- **Create a complex data model - ASP.NET MVC with EF Core**
URL: <https://docs.microsoft.com/en-us/aspnet/core/data/ef-mvc/complex-data-model?view=aspnetcore-5.0>
Sidst set 31-05-2021
- **Create Add sorting, filtering, and paging - ASP.NET MVC with EF Core**
URL: <https://docs.microsoft.com/en-us/aspnet/core/data/ef-mvc/sort-filter-page?view=aspnetcore-5.0>
Sidst set 31-05-2021
- **Multi-factor authentication in ASP.NET Core**
URL: <https://docs.microsoft.com/en-us/aspnet/core/security/authentication/mfa?view=aspnetcore-5.0>
Sidst set 31-05-2021
- **Ofte stillede spørgsmål om den gratis Azure-konto**
URL: <https://azure.microsoft.com/da-dk/free/free-account-faq/>
Sidst set 31-05-2021
- **What is DevOps?**
URL: <https://docs.microsoft.com/en-us/devops/what-is-devops>
Sidst set 31-05-2021
- **What is Continuous Integration?**
URL: <https://docs.microsoft.com/en-us/devops/develop/what-is-continuous-integration>
Sidst set 31-05-2021

Bøger:

- Price, Mark J.: **Beginning C# 8.0 and .NET Core 3.0 – Modern Cross-Platform Development**. Fourth Edition: Packt Publishing 2019
- Smith, Steve “ardalis”: **Architecting Modern Web Applications with ASP.NET Core and Microsoft Azure ([AMWA])**. EDITION v3.1: Microsoft Corporation 2020