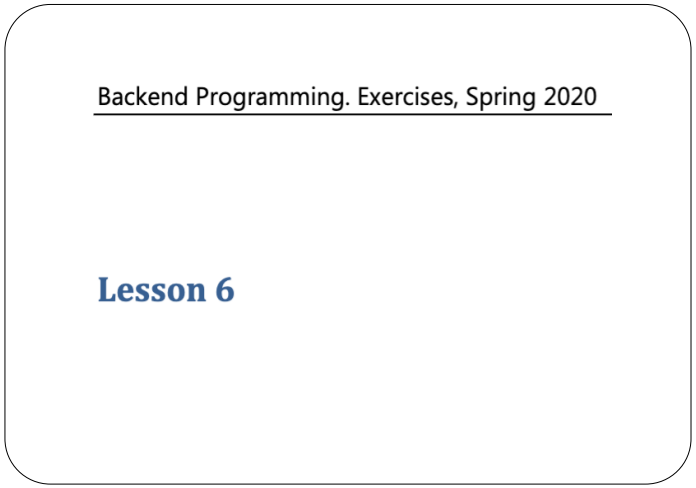


MANDATORY 2 RAPPORT

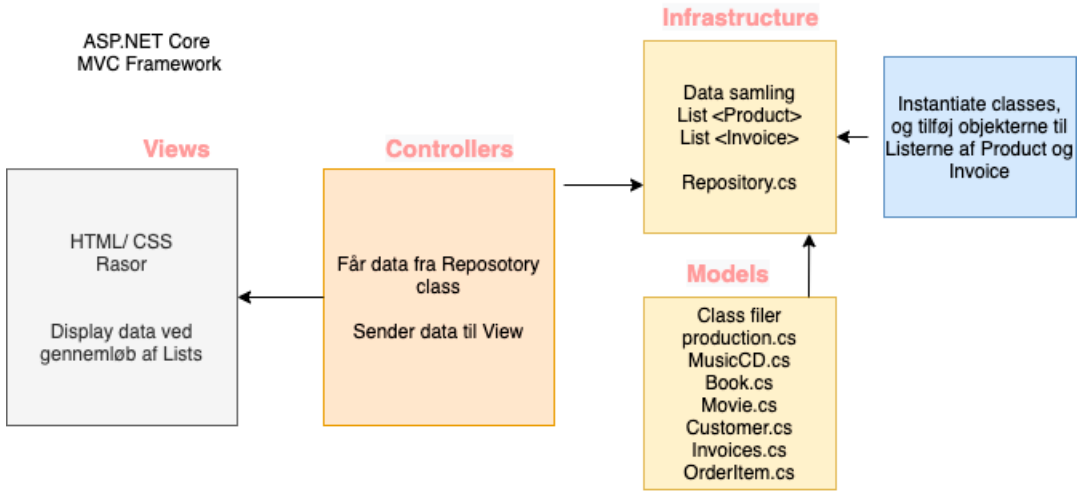
https://mbmstore.prfrankild.dk/

MVC architecture

ASP.NET core MVC App er et Microsoft open source software som kan bruges til at udvikle objektorienteret http / https platformes kodning, hovedsageligt med C# som programmeringssprog, men også andre, Visuel, base, J#. Der er indbygget Kode Editor som kan minimere code og hjælpe med code intellisense programmering generering, som gør kodningen meget lettere. Den gratis version hedder Visual Studio Community Edition.[1]

ASP.NET har en arkitektur, hvor brugeren anmoder om en http-adresse, den rammer en router der finder den rigtige controller med tilhørende model, som derefter beregner indholdet i en skabelon visning. Her html genererer et output til den anmodende klient gennem den skabelon, der er aktiveret fra controlleren.[2]

ASP.NET core framework har en central placeret editor under tabs af åbne vinduer og med en værktøjslinje over den. Under editoren er der en error blok hvor det kan ses om der under kodning og build af projektet registreres fejl. Et andet blok valg er Solutions Explorer hvor der er tilgang til mappestrukturen i projektet.



1 Billede af Visual Studio Community Edition. Et Microsoft open source software.

2 Visuel Studio er udvikle til objektorienteret programering fra Controller af objecter til Views.

Når du starter et nyt projekt og har valgt at det er et ASP.NET core web app projekt, bliver der gjort adgang til en application struktur som indeholder adgang til ekstern Connected Service og Dependencies, som er en cloud service. Wwwroot mappen er stedet hvor filer med særlige struktur i applikationen bliver placeret sammen med libman.json filen som kan downloade css og javaScript filer per session lokalt.[3]

Properties, som er særlige indstillinger i projektet. Program og Startup som styrer starten og konfiguration af objekt programmeringen til Index View filer ved bla. at bruge endpoints.MapControllerRoute(name: "default", pattern: "{controller=Home}/{action=Index}/{id?}"); });

.

Det er i Startup filen at MVC Konfigurationen kaldes. Det gøres ved at der opsættes en Get method,

ved brug af

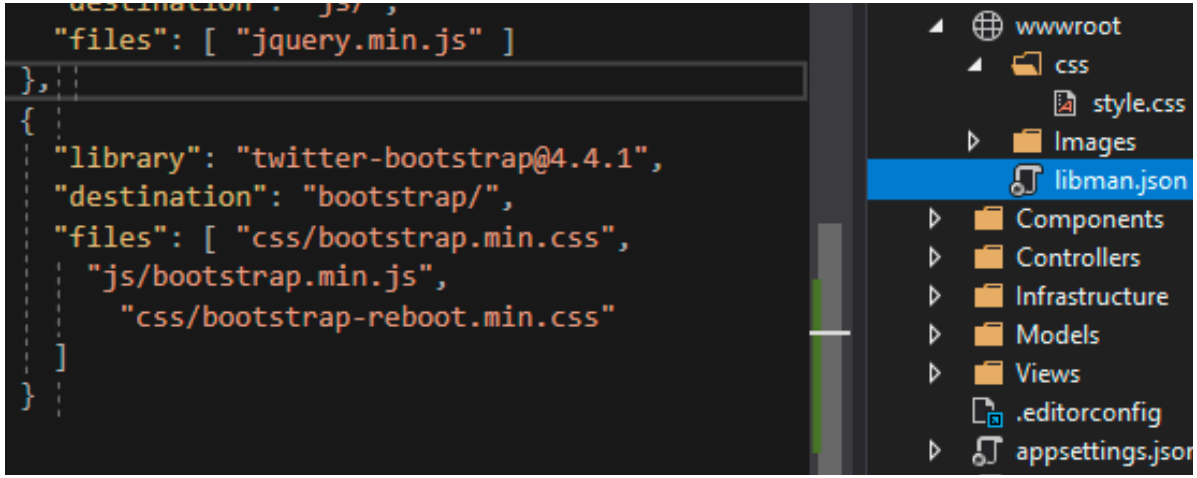
ConfigurationService.[4]
service.AddControllerWithViews

MVC er betegnelsen for en mappestruktur, hvor der genereres automatiske koblede class filer fra mapperne Controllers, Models og Views. Det sker ved at der højreklikkes på mappen og tilføjes en Class med navn. Et View kan have forskellige referencer

@addTagHelper hjælper med link action i Views,

@using MbmStore.Models;
@using MbmStore.Infrastructure;
som refererer til classens placering,

@model Product
@model IEnumerable<Product>
Gør View til strongly typed,



3 Libman.json fil med kode til at hente javaScript og bootstrap, fra hvilket library, hvortil er destinationen og files, filer som skal hentes.
4 I Startup filen skal MVC konfigureres, med Configuration, ConfigurationService, endpoints og home Index id?.

@{Layout = "~/Views/Shared/_Layout.cshtml";}
viser layout default fil,

@model MbmStore.Models.ViewModels.ProductsListViewModel
trækker data fra ProductsListViewModel classen,

@{ViewData["Title"] = "Products"; }
Giver navnet på produkt object.

Controlleren kan også have data skrevet direkte i Index, som kan bruges i et udviklingsmiljø, som hjælp til asp strukturen.

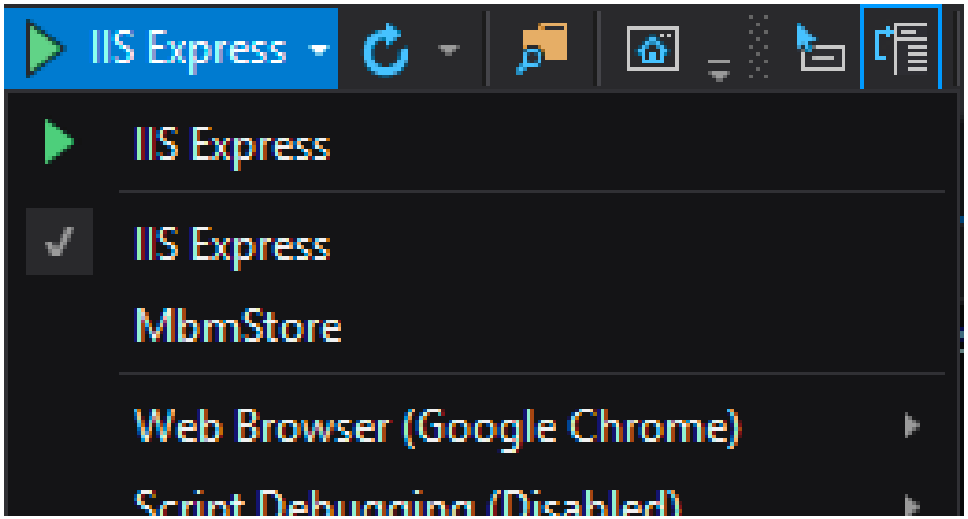
```
using System.Linq;
using Microsoft.AspNetCore.Mvc;
using MbmStore.Infrastructure;
using MbmStore.Models.ViewModels;
```

Herefter er controller filen opdelt i lag fra projekt navn "Namespace", herefter class navn "HomeController", indhold med "index" og en "return view" som returnerer et resultat fra index.

Det er samme fremgangsmåde når man danner en View fil, hvor man kan vælge et layout template som ligger i en Shared mappe[5], hvor flere templates kan tilføjes. View bliver skabt af dens controller fil som kender data placeringen.

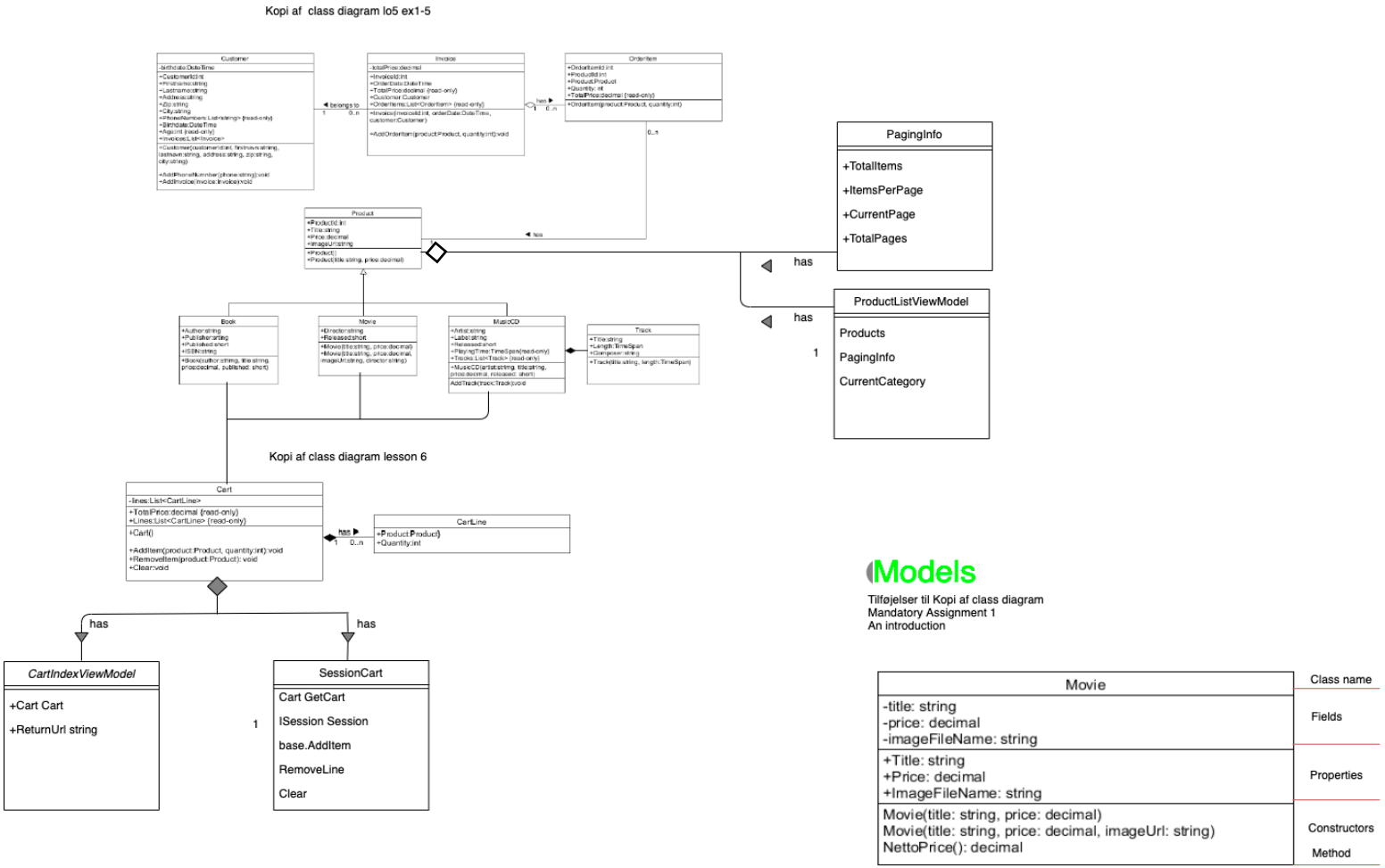
Data fra Controlleren kan blive vist ved brug af viewData som er et "dictionary object", ViewBag som er dynamisk eller strongly typed view som har intellicence.

View kan bliver vist på en localhost server IIS Express, som er del af Visual Studio.



5 _layout.cshtml i Shared mappen er default template af contents delen af View, som kan outputtes lokalt af ISS serveren.

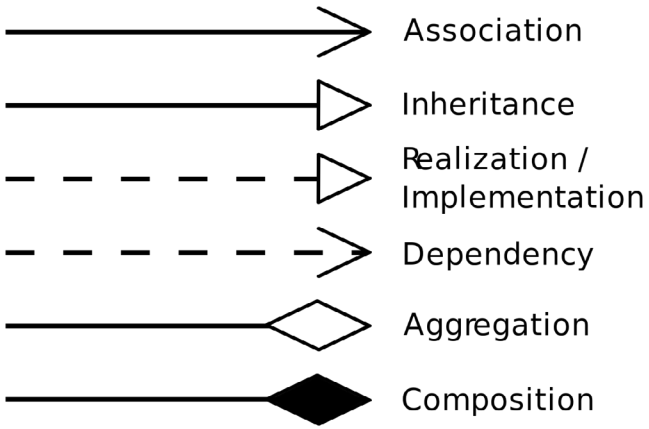
6



6 Et UML class diagram indholdt Models classes fra MbmStore projectet, hvor Cart er afhængig af CartIndexView og SessionCart.

Ved brug af UML class diagram[7] er der flere måder at forklare relationen mellem de forskellige class filer.

Begge klasser kender foreningen - solid linje mellem uden pile.



En måde:
Kun en klasse ved om den anden - Åbent pilhoved.

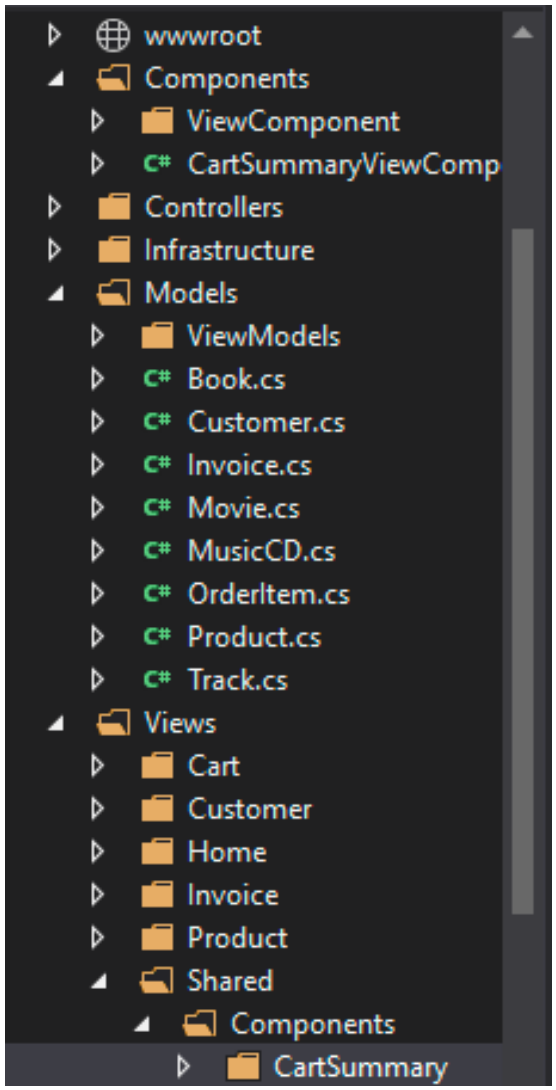
Afhængighed:
Når ændringer i en klasse forårsager ændringer i en anden klasse - Stiplet linje.

Aggregering:
Class indeholder flere forekomster af en anden class - Hul diamant.

Komposition:
Slettes container Slettes class. - Solid diamant.

7 UML class diagrams relationen af class filer, hvor diamant er container mellem flere classes.

8



Strongly Typed View

Strongly Typed view binder den refererede class til View gennem at referere til @model object og IEnumerable for at liste et object som bruges i View. Det kan sættes op med et the _ViewImports.cshtml File in the Views Folder med @using Razor.Models, som gør at der ikke skal tilføjes Razor eller Models i andre Views. Ex.
@using MbmStore.Models;
@model Products
@model IEnumerable <Invoice>
Dette giver mulighed for IntelliSense og fejlstavning, når der er skrevet @Model..som kaldenavn model står i stedet for Invoice:s

@Model.Id
@Model.Title

Tag Helpers

8 MbmStore som det ses i Solution Explor fra toppen wwwroot, Components, Controller, Infrastructure, Models, View, Shared.

Tag Helpers er en ny opsætning i ASP.NET Core som hjælpen html Viewet med forskellige egenskaber som er typisk er prefixed af asp tag tilføjet den normale html. Det er en serverside hjælp til html rendering bla. Links, Label, Form, Form elements, Image, Anchor, Validation messages (ved Span elementer).

Tag Helpers virker ved at der indsættes en linje i toppen af View som kan have forskellige funktioner @addTagHelper*,Microsoft.AspNetCore.Mvc.TagHelpers, som fx. kan bruges som output data, i form af asp kode <form> <input class=”form-controllller” asp-for=”title”/> eller
Model: public string Email { get; set; }
Tag helper: <label asp-for="Email">Please enter your Email</label>
Html out<label for="Email">Please enter your email</label>

Tag helpers kan bruges i forbindelse med tal (int), decimaler (double), tekst (string) og en masse andre C# termer inklusiv hidden input data, passwords, phone, email, url og til at give en select mulighed af en liste i en dropdown i View.

I toppen binding til Book objectet: @model MbmStore.Product.Book. Dropdown i html: <select asp-for="Title"asp-items="ViewBag.TitleList"></select>.

Binding

Model binding udtrækker værdier fra en anmodning og bruger dem til at oprette .NET-objekter, der sendes som metode parametre til den handlingsmetode, der udføres af Models. fx.

public string Index1(int id) {return id.ToString();}

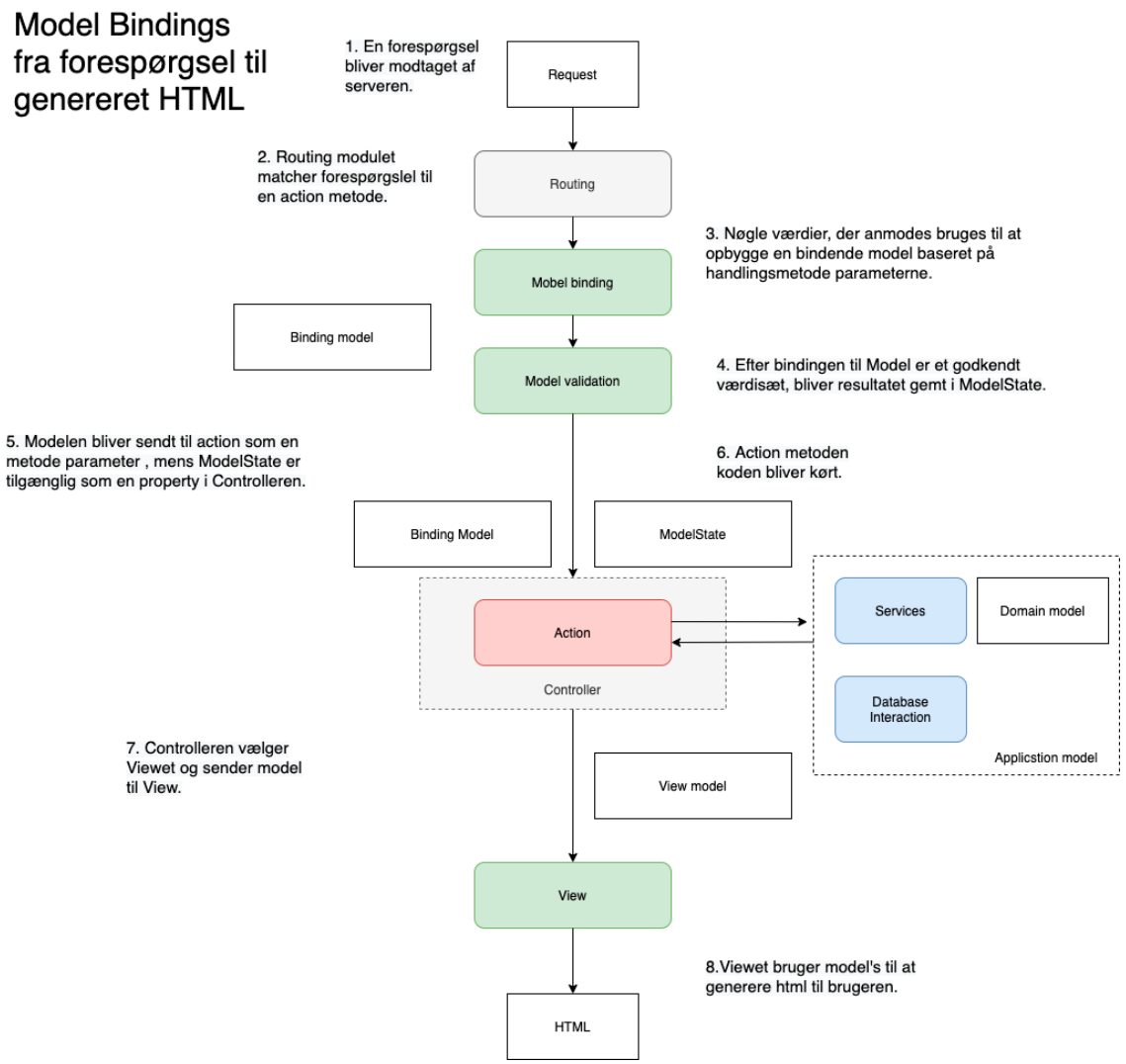
Der kan også Bindes fra komplekse form fx.

[HttpPost]
public ActionResult Index(Person p) {return View(p);}

Fordelen er at der gennem Binding kortlægning indgående data til et objekt som kan blive et gentaget data kortlægning. Det betyder at Binding ofte ses ved sammenligning i kode. Det er synligt og med til at mindske fejl i kodningen.

Binding med form values tilføjer data fra brugeren i html, som gennem Linq kan bruges til at skabe et View. Route values bindes ved at data trækkes fra en database.

Model binding bruger en refleksion af hvordan indholdet af en classe ser ud iform af enhed = new enhed(,,)



9 Mobel binding med validering, action controller, services, interaction, view.

Binding med form values tilføjer data fra brugeren i html, som gennem Linq kan bruges til at skabe et View. Route values bindes ved at data trækkes fra en database.

Routing

Der er i projektet tale om attribute routing som gør det muligt at specificere routing mellem Controllers og Action methods.

Valid Route Patterns URL fx. mysite/{username}/{action} eller public/blog/{controller}-{action}/{postId}.

Der eksistere også Convention based Routing, hvor der er et separat rute systemet, ved at Controllers ikke kender eller har er afhængighed til routing konfigurationen. Alt info om routing findes i en fil.

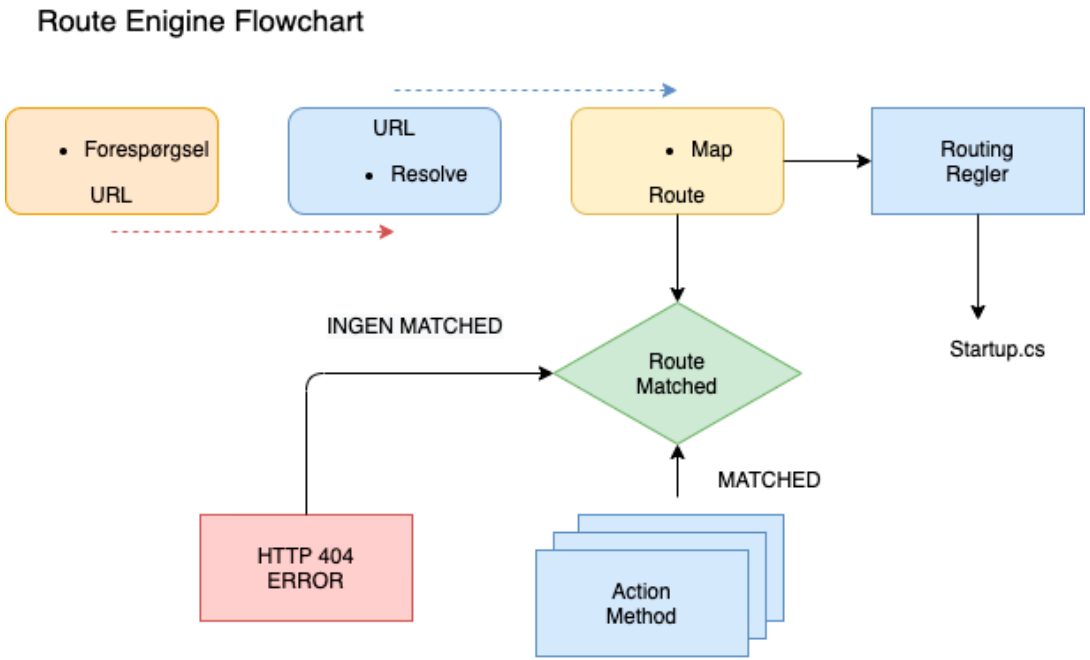
De to former for routing virker i samme projekt, men attribute routing har forret over convention based routing

Den mest brugte opsætning af URL routing:

Home (default);

host/Product
host/Product/Book

Rute systemet matcher den sekvens, hvor de startende med:
De mest specifikke ruter skal defineres først. De mest generelle ruter skal defineres sidst.[10]



10 Billed af routing flowchart fra forespørgsel til action method eller error - HTTP 404.

MbmStore projekt

Vi står med et projekt, som er sat op med en Infrastructure mappe med filen Reposotory. Reposotory danner lister fra Customer, Invoice og Product objects.

Customer existere som en list af objects som kan trækkes ud i et View og bruges til at fylde data i fx invoice. Invoice er sin egen liste og bliver fyldt op af data fra customer og OrderItem. OrderItem har et Quantitiv * product id som bliver tilføjet til Invoice.

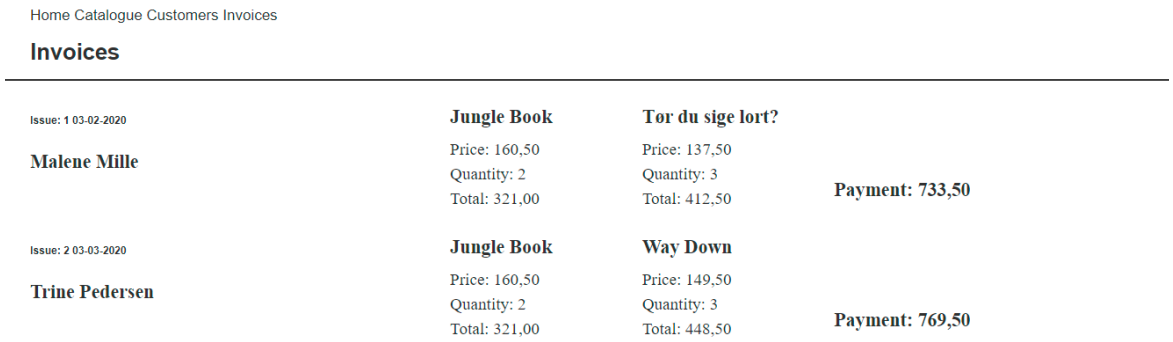
Product er delt i tre object typer Book, Movie og MusicCD, som udover object navn har en Track list, som bliver tilføjet til hvilket MusicCD object den tilhører, ved brug af Add.

Der er Controllers class til Home, Customer, Invoice og Product som return data til View class html som outputtes i en url fx. https://localhost:44337/(Product/).

Dette gør en outgoing URL til en valid url af et ekstern link
<a asp-controller="Home" asp-action="Index"
asp-protocol="https"
asp-host="myserver.mydomain.com">

Sådan ser en outgoing URL ud

.



Udvalgte senarier i MbmStore

MbmStore er delvis på plads og enkelte filer bliver undervejs ændret i form af nye funktioner og ved at der opstår mulighed for reducering af kode. Det er væsentlig for yderlig forståelse af binding og routing.

Invoice er et foreach template af hver invoice ud fra Costumerld, som så bliver outputtet i html, men for at kunne vælge mellem og kun forvise den valgte invoice fra en dropdown skal der skrives noget aps kode ind i starten af invoice View. Det kræver at Customer har en automatic properties for Custormerld som skal virke som en unik value, henvisning i dropdown ved at eksistere i hver Invoice i Reposotory. For at gøre det muligt skal der genereres en liste i InvoiceControlleren fra Reposotory.

```
// the InvoiceController class
0 references
public class InvoiceController : Controller {
    private List<Invoice> invoices;

    0 references
    public IActionResult Index() {

        // declare the list
        List<SelectListItem> customers = new List<SelectListItem>();

        // generate the dropdown list
        foreach (Invoice invoice in Repository.Invoices)
        {
            customers.Add(new SelectListItem
            {
                Text = invoice.Customer.Firstname + " "
                + invoice.Customer.Lastname,
                Value = invoice.Customer.CustomerId.ToString()
            });
        }
```

I namespace tilføjes:
using MbmStore.Models;
using Microsoft.AspNetCore.Mvc.
Rendering;
using System.Collections.Generic;
using System.Linq;[11]

11 Linq giver mulighed for smart kodning i kendte kode systemer fx. dropdown, med simpel kodning af foreach af SelectListItem.

Linq kan bruges til at fjerne dublicates fra listen med samme id. En simpel kode kan gøre det:

```
customers = customers.GroupBy(x => x.Value).Select(y => y.First()).OrderBy(z => z.Text).ToList<SelectListItem>();
```

GroupBy bruger kun første element, så objektet ikke gentages sig ved flere invoices til samme kunde. For at sende listen til ViewData med den key Custormerld der er skabt kræver det:

```
ViewData["Customers"]= customers;
```

Som gør dropdown i Viet ved form select:

```
asp -items="@ViewData["Customers"]as List
```

Ved brug af et filter som Customer selected, skal en invoic vises ved klik, det sker inde i Controlleren med en overload function Index som rammer en httppost layout:

```
 [HttpPost]public IActionResult Index(int?customer){}
```

I Invoice Controlleren skal der nu sætte et filter af invoices baseret på Custonerld ved brug af linq:

```
if(Customer !=null) {
invoices =Repository.Invoices.Where(r => r.Customer.Customerld==customer).ToList();;}
```

og videre til View Invoices, som også er en binding:

```
return View(invoices);
```

Det er muligt at ændre den customer der vises i dropdown efter klik ved at tilføje en else statement til den gamle som så skal være en if:

```
else {customers.Add(new SelectListItem{Text = invoice.Customer.Firstname +" " + invoice.Customer.Lastname, Value = invoice.Customer.Customerld.ToString()}
```
[12]

12 En html razor view af en dropdown i Invoice MbmStore, hvor navnet er det samme i formen og i viewet.

For at ændre Product Viewer til strongly typed view, ændre vi Controlleren så den sender data som strongly typed Model view:

```
public IActionResult Index()
{
return View( Repository.Products);
}
```

Herefter kan vi ændre view til et Strongly type view:

```
@model IEnumerable<Product>
@{
ViewData["Title"]="Producs";
}
```

For lige at tage den igen giver det kortere kodning og intelisencer i AST.NET Core, hvilket betyder at hvis vi skal trække en liste af categorier, vil ASP.NET iterate model af categorier for at finde dem som matcher.

skal der tilføres en cart er det nødvendigt ikke at bruge opsætningen af prisen i Invoice class og Orederlten class, men oprette en ny ViewModels mappe inden i models mappen for de nye classes Cart, CartLine i en eller to filer. De minder om Orderltem i Invoice. Der skal være en "remove Product" og en "Continue shopping".

For at gøre cart funktionel skal der være en add Product knap på hver product i Product (Catalogue).

Her skal der bruges Tag Helpers. Der skal en UrlExtensions.cs class til Infrastructure mappen og den skal definere en extension method PathAndQuery til class navnet HttpRequest.

Nu skal der lægges en knap i product View classen i en top og bund i den færdige udgave hvor de enkelte funktioner i View er delt op i classes for at minimere kodning. [13]

```
@model MbmStore.Models.Product
@using MbmStore.Infrastructure
@addTagHelper *, Microsoft.AspNetCore.Mvc.TagHelpers

@addTagHelper *, AuthoringTagHelpers
<div>
    <form id="@Model.ProductID" asp-action="AddToCart"
          asp-controller="Cart" method="post">
        <input type="hidden" name="ProductID" value="@Model.ProductID" />
        <input type="hidden" name="returnUrl" value="@ViewContext.HttpContext.Request.PathAndQuery()" />
        <span class="card-text p-1">
            <button type="submit"
                    class="btn btn-success btn-sm pull-right" style="float:right">
                Add To Cart
            </button>
        </span>
    </form>
</div>
```

13 Billede af Add to Cart kodningen i _AddToCartButtom.cshtml, med ProductID og returnURL af Request.

Knappen ligger for sig i en Component mappe i shared mappen, som stadig indeholder Layout og andre template layout filer.

For ikke at miste data fra add Product knappen i product kræves der nogle ekstra service koder i startup filen ConfigureServices:

services.AddMemoryCache();
services.AddSession();

og i Configure:

app.UseSession();.

Der skal tilføjes en CartController og den kode kræver den indeholder ASP.NET session state muligheder for at gemme og udtrække Cart objects og middleware som bruger cookies eller URL som sammensætter flere handlinger fra samme session af samme Customerld som gør samme single session. vi vil have at der ikke bliver konflikter mellem brugere af vores Cart system og at alle detaljer slettes efter manglende køb over tid.

For at kunne arbejde med flere køb i samme cart og kunne få brugeren til at gå til og fra Cart skal der være en "Defining Session State Extension Methods" som gør at data fra session interface bliver gemt i en json.net package til brug i en serie køb Session Cart objects. Det gør vi ved at lave en fil i infrastructure mappen, som hedder SessionExtensions.cs.[14]

det gør det nemt at trække form data frem igen ved at tilføje denne statement til Controller:

HttpContext.Session.SetJson("Cart","cart");

For at trække de samme data bruges:

Cart cart = HttpContext.Session.GetJson<Cart>("Cart");

Sammen med en zero parameter i Product class:

public Product(){}.

```
public static class SessionExtensions {
    2 references
    public static void SetJson(this ISession session, string key, object value) {
        session.SetString(key, JsonConvert.SerializeObject(value));
    }
    1 reference
    public static T GetJson<T>(this ISession session, string key) {
        var sessionData = session.GetString(key);
        return sessionData == null ? default(T) : JsonConvert.DeserializeObject<T>(sessionData);
    }
}
```

14 Billede af kodningen i SessionExtentions i Infrastructure mappen, med SetJson og getJson, som returnere sessionData.

For at få vist indhold i cart og få mulighed for at fjerne eller forsætte handel skal der i Models/ViewModels laves en ny class CartIndexViewModels.cs med automatiske properties Cart Cart og ReturnURL sammen med action method i CartController class properties.[15]

```
public ViewResult Index(string returnUrl)
    return View(new CartIndexViewModel
    {
        Cart = cart,
    ReturnUrl = returnUrl
    });
}
```

For at gøre Cart synlig skal der være en Cart mappe med Cart.cshtml fil i Views hvor der skal være en binding i toppen til:

@using MbmStore.Models.ViewsModels,
@modelnCartIndexViewModel
@addTagHelper*,Microsoft.AspNetCore.Mvc.TagHelpers.

I View skal der være en foreach (var line in Model.Cart.Lines) statement med følgende indhold:

```
@foreach (var line in Model.Cart.Lines)
{
    <tr>
        <td class="text-center">@line.Quantity</td>
        <td class="text-left">@line.Product.Title</td>
        <td class="text-right">@line.Product.Price.ToString("c")</td>
        <td class="text-right">
            @((line.Quantity * line.Product.Price).ToString("c"))
        </td>
        <td> <form asp-action="RemoveFromCart" method="post">
            <input type="hidden" name="ProductID" value="@line.Product.ProductID" />
            <input type="hidden" name="returnUrl" value="@Model.ReturnUrl" />
            <button type="submit" class="btn btn-sm btn-danger"> Remove </button> </form> </td>
    </tr>
}
```

En ComputeTotalValue() og en Href til ReturnURL knappen.

Der er sat en remove function op i CartController som nu bare skal tilføjes hvert produkt i cart View.[16]

Det gøres ved at bruge en form med asp-action method i foreach looped.

```
<td> <form asp-action="RemoveFromCart" method="post">
    <input type="hidden" name="ProductID" value="@line.Product.ProductID" />
    <input type="hidden" name="returnUrl" value="@Model.ReturnUrl" />
    <button type="submit" class="btn btn-sm btn-danger"> Remove </button> </form> </td>
```

15 Billede af implementering af Cart Controller, med return view af Cart og ReturnURL.
16 Billede af form asp RemoveFromCart, med input hidden ved button submit.

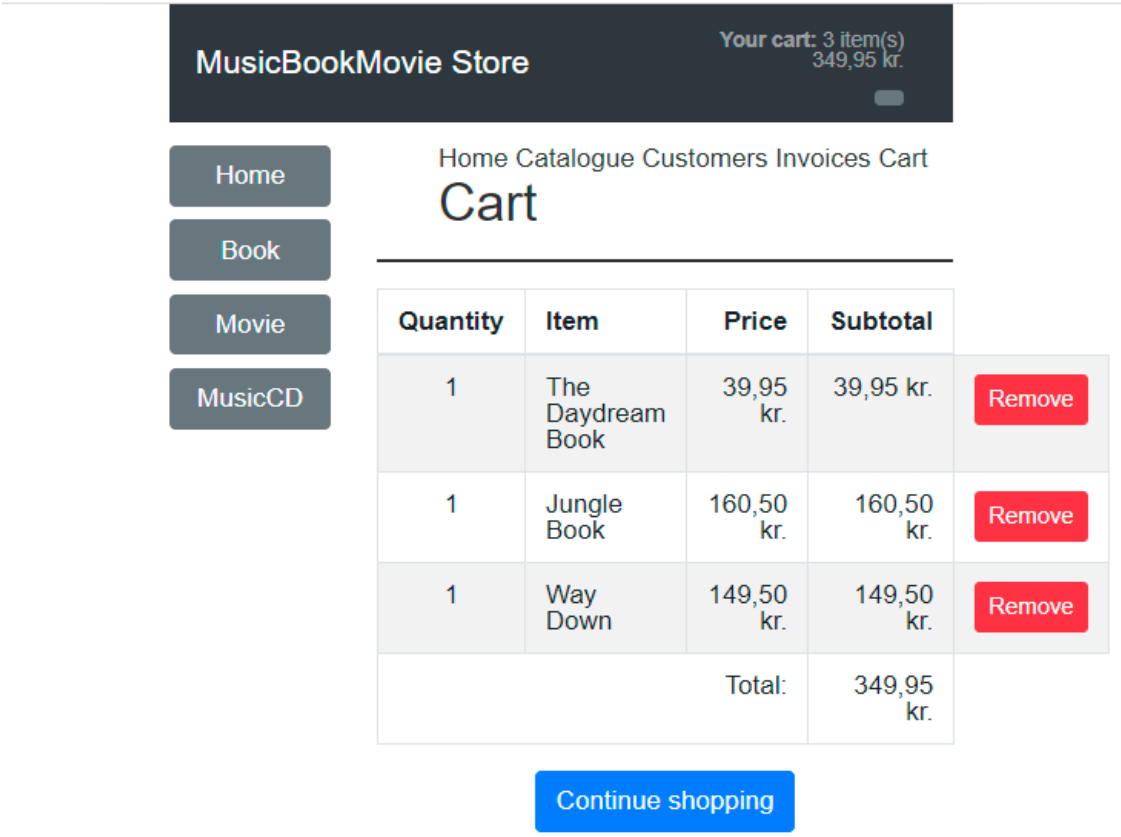
Startup.cs skal bindes til:
using MbmMovie.Models.ViewModels;
using Microsoft.AspNetCore.Http;

med tilføjelser til ConfigurationService:
services.AddScoped<Cart>(sp => SessionCart.GetCart(sp));
services.AddSingleton<IHttpContextAccessor, HttpContextAccessor>();

Nu er der opsat en handels funktion til Product Repository med knap til at tilføje produkt, et View til at outputte til html af køb og totalpris, med mulighed for at fjerne et valgt produkt og vende tilbage til Product for yderligere køb i samme session. Der kan tilføjes mange andre funktioner til en cart og der kan reduceres i kodestykker som gentages på flere classes.

Konklusion

Der har været flere problemer med at få en virkelig smart bruger den objektorienteret strukur, som er muligt i ASP.NET Core. Tags helpers har været uklart helt til slut i opgaven, men har en afgørende effekt i måden projektet fungere. Shared filer fungere og optimere processer og tid for afviklingen. Jeg vil arbejde videre og har fået en god start med at finde løsninger i Pro ASP.NET Core MVC af Adam Freeman.



Kildehenvisninger:

https://docs.microsoft.com/
https://github.com/eaaaweb/

Pro ASP.NET Core MVC 2 Seventh Edition (2017)