



PRODUKTRAPPORT

Deltager: nicolai søndergaard aagaard, Projektitel: TheOnlineMarket.dk



11. NOVEMBER 2024-06. DECEMBER 2024

TheOnlineMarket.dk
Skole: Mercantec, Viborg
Vejleder: Henrik Thomson

Indhold

Definition af produktet	3
Kravspecifikation	4
Krav	4
Skærm billeder	6
Hjemmeside	7
Sikkerhed	15
Adgangsret for brugere.....	15
sikkerhedsroller.....	15
Projekt dokumentation	15
Arkitektur	15
Præsentationslaget:.....	15
Applikationslaget:	15
Forretningslaget:.....	16
Dataadgangslaget:	16
Arkitektur og Design	16
System arkitektur:	16
Backend	16
Frontend.....	17
Database.....	18
Formål	19
Relationer	20
Implementeringsdetaljer	21
Backend.....	21
Dependency injection	21
Konvertering mellem modeller	21
DatabaseLayer	23
Models	23
Endpoints.....	23
Object-relational mapping (ORM)	24
Frontend hjemmeside	25
Indledning.....	25
Forms	25
Modals.....	25

Toastr.....	27
Eksterne biblioteker	27
Strukturerung af Webservice og webservice-kald.....	27
Authentication	28
Kandidater	29
Frontend.....	29
Backend	31
Server.....	32
Diagrammer	33
Server.....	33
Brugernavn:.....	33
Password:	33
Forbindelses streng	34
Linket	34
links	35
Github repositories.....	35
Frontend.....	35
Backend	35
Trello.....	35
Design	35

Definition af produktet

TheOnlineMarket.dk er en Market Place for at sælge og købe produkter fra den private bruger, med brug af Ai og dens store muligheder, der gives der kraftige værktøjer til bruger for at finde det rigtige produkt de vil købe, samt hjælpe dem der er usikker på hvad de kan sælge deres gamle ting for, selvfølgelig så gives der mulighed for at live chatte med køber/sælger for at let kunne finde ud af hvorhenne og hvordan de skal betale for produktet.

Dette projekt er delt op i 2 faser, første fase er til svendeprøven og anden fase er ekstra jeg kan, men som ikke er en del af svendeprøven.

Kravspecifikation

Krav

Funktionelle Krav:

forside

- se alle produkter på forsiden
 - se billede og pris af produkter
 - kan se kategorier
 - prisinterval-filter for søgninger
 - søgning efter fx iphone
 - semantic search (Fase 2)
 - reportere annonce (Fase 2)
- Navigationsbar
 - Profil billede man kan tryk på
 - Log ind hvis man ik er
 - Register hvis man ik er
 - Hvis man er logget ind
 - Viser den ens navn
 - Kan gå ind i side
 - Fremviser købte produkter hvor man så kan give en rating og feedback på sælgeren (Fase 2)
 - hvis man har fået tilladelse til at sælge
 - vil man få en knap der hedder produkter hvor man kan gå ind på en ny side hvor man kan oprette en ny annonce samt se alle ens egen annoncer
 - du vil også kunne redigere i alle dine annoncer
 - du vil også kunne fjerne dine annoncer
 - ved oprettelse af en ny annonce
 - indsætte et link fra facebook market place eller dba (den blå avis) for at gøre det lettere for en sælger
 - vil man kunne få en Ai til at give pris anbefalinger (Fase 2)
 - hvis man har admin eller moderator tilladelser (Fase 2)
 - kan man se en knap til for at gå ind på admin/moderator side (Fase 2)
 - En log af
 - Indstillinger
 - Anbefalede
 - Åbner en ny side
 - Viser anbefalede produkter ud fra tidligere søgninger og køb (Fase 2)

- Prisændringer
 - Viser prisændringer i produkter man er interesseret i
- Notifikationsknap
 - Viser seneste og ikke læst
 - Giver besked om prisændringer på produkter man er interesseret i
 - Se hvorfor et produkt er fjernet af en admin/mod (Fase 2)
- Chat beskeder (Fase 2)
 - Vise modtaget beskeder (Fase 2)
- Man skal kunne få notifikationer
 - Om prisfald i produkter man er interesseret i
 - Om ny chat besked (Fase 2)
- Indstillinger
 - Kan du registrere og betaler for at blive en sælger
 - Du kan se og ændre på dit profilbillede
 - Se og ændring af beboelsesområde
 - Se og ændring af Mail
 - Se og ændre kreditkort
 - Viser ens navn og efternavn
- Individuel produkt side
 - Forstørret billede af produktet
 - Navn på produktet
 - Price på produktet
 - Hvornår det blev lagt op
 - Beskrivelse
 - Område den er i
 - Tilføjelse til liste af interesseret produkter
 - Fornavn på sælger
 - Skrev til sælger (Fase 2)
 - Reportere annonce (Fase 2)
- Admin/moderator side (Fase 2)
 - Kan se reporterede annoncer
 - Samt slettede de reporterede annoncer hvis der er noget galt med dem og give en strike, hvis det er alvorlig nok (Fase 2)
 - Admin kan også fjerne en moderator, men en moderator kan ikke fjerne en admin eller en anden moderator (Fase 2)
 - Sendte en notifikation om at man har fjernet et produkt til ejeren af produktet (Fase 2)

Ikke Funktionelle Krav:

- Sikkerhedsforanstaltninger:
 Den primær sikkerhed som giver sikkerhed vil være apien, da man i en hjemmeside ikke kan stoppe folk for at inject javascript kode ind, derved er det vigtigste på apien af man har authorize og authentication, som jeg også vil gøre

brug af. En god måde på at stoppe almen bruger, er at gemme router til sider, de ikke har adgang til at gemme knapperne til de router de ikke har adgang til.

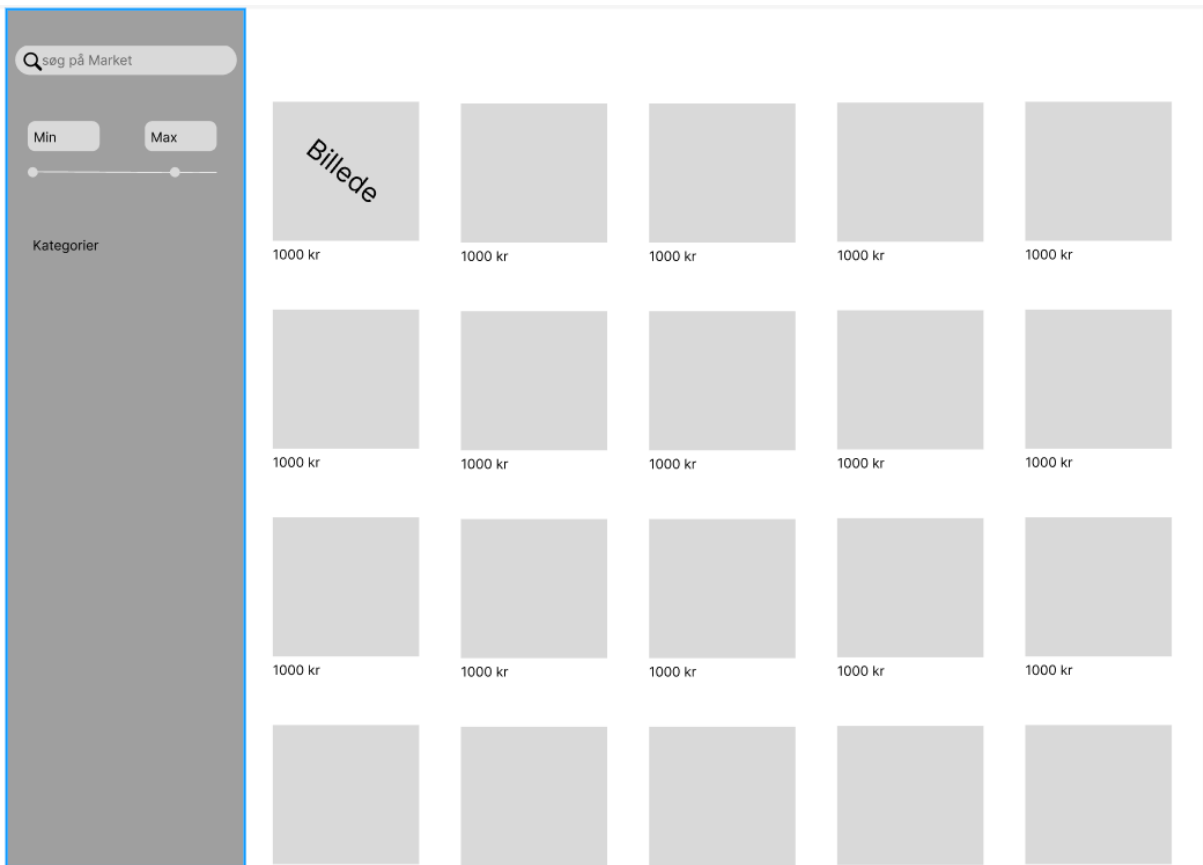
- Brugervenlighed:

Der er tænker om at den vil være intuitiv ved hvad af designet, der er karakteristiske træk fra Facebook Marketplace platform, derved kan mange hvis ikke alle alder der må gøre brug af det kunne gøre brug af det

Skærm billeder

Brugere skal kunne se data svarende til følgende skærm billeder.

Hjemmeside



Forside.



Navigationsbar



billeder



Titel der kan fylde en god del med nok info for en titel

Pris fx 10000 kr. +mere

dato det blev slået op

Gem i liste

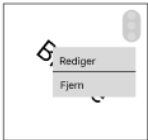








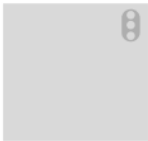
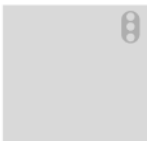


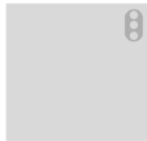
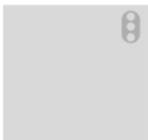
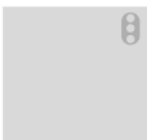

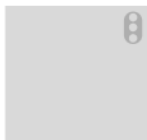

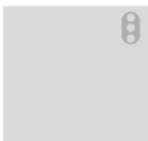
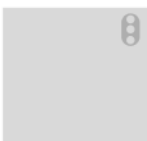
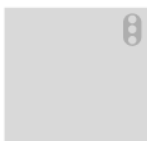
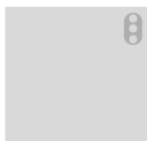
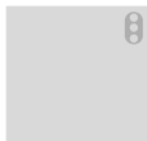





detaljer

her kan der stå alt mulig info fx hvis det var en bil, kunne man skrive det er en mitsubishi lancer 2.0i fra 2006
Dansk afgift
har kørt x kilometer
har x, y, z, u, t og mere ekstra udstyr og meget, meget mere

i omregn af "By"

Sælges af "Fornavn"

viewItem side

					<div>Brug Link</div> <hr/> <div>Titel</div> <div>Pris</div> <div>Beskrivelse</div> <div>tilføj billeder</div> <div>     </div> <div>Tilføj announce</div>
1000 kr	1000 kr	1000 kr	1000 kr	1000 kr	
					
1000 kr	1000 kr	1000 kr	1000 kr	1000 kr	
					
1000 kr	1000 kr	1000 kr	1000 kr	1000 kr	
					
1000 kr	1000 kr	1000 kr	1000 kr	1000 kr	
					

Announce side

indstillinger

Profil

Kredit Kort

Gemte vare

Profil billede

Fornavn

EfterNavn

Fødselsdato

By

Postnummer

adresse og vejnummer

mail adresse

Opdater profil

Indstillings side inde i profil

indstillinger

[Profil](#)

[Kredit Kort](#)

[Gemte vare](#)

for 10 Kr. vil du kunne lægge announcer
op og sælge ting, du skal bare registrere
dit kredit kort

Card holder full name

Card Number

Expiry date/ov

CVV

Register

Indstillings side inde i kredit kort inden man har tilmeldt et kreditkort

indstillinger

Profil

Kredit Kort

Gemte vare

du har registreret dig til at kunne sælge

Navn

xxxx 0560

Card holder full name

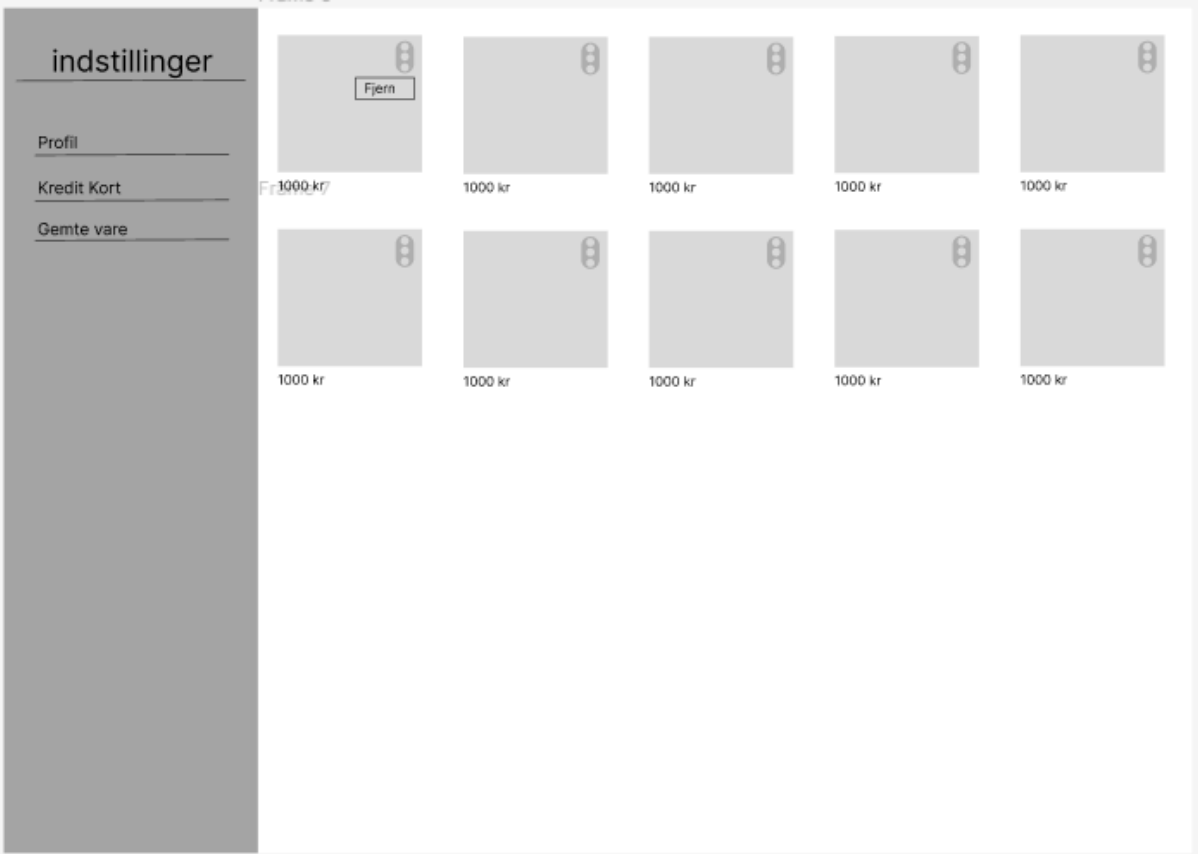
Card Number

Expiry date/love

CVV

Register New

Indstillings side inde i kredit kort efter man har tilmeldt et kreditkort



Indstillings side inde i gemte vare

register

Fornavn

Efternavn

Mail

Password

Password

Fødselsdato

By

Postnummer

Adresse og vejnummer

Tilføj billede

Register

Register side

Welcome Again!

Mail

Password

Forgot Password?

login

Login side

Sikkerhed

Adgangsret for brugere

1. Online Market vil kræve, at hver bruger har en unik identifikator i form af en e-mailadresse,
2. Noget af hjemmesiden skal være låst bag en loginprocess, hvor brugeren skal indtaste en gyldig kombination af en e-mailadresse og adgangskode for at få adgang
3. noget af hjemmesiden skal være låst bag en rolle, hvor brugeren skal have en gyldig rolle for at kunne få adgang
4. adgangskoden skal minimum bestå af 8 karakter, indeholde mindst et specialtegn og mindst et stort bogstav

sikkerhedsroller

projektet har de har 4 roller med mulighed for udvidelse i fremtiden

1. bruger
2. sælger
3. moderator (fase 2)
4. admin (fase 2)

Projekt dokumentation

Arkitektur

Projektet er opdelt i 4 lag, præsentationslaget, applikationslaget, forretningslaget og dataadgangslaget, det er for at skabe klarhed og adskillelse af ansvar. Hvert lag har en specifik rolle og en klar opgave, som gør det let at forstå, vedligeholde og udvikle videre på.

Præsentationslaget:

Dette lag repræsenterer brugergrænsefladen og er ansvarligt for at præsentere data og funktionalitet på en måde, der er forståelig og brugervenlig for slutbrugeren. Det håndterer visning af information, som brugeren kan se og interagere med, og fungerer som forbindelsen mellem brugeren og applikationens underliggende logik.

Applikationslaget:

Dette lag fungerer som mellemlid mellem præsentationslaget og forretningslaget. Det er ansvarligt for at håndtere forretningslogikken og implementere funktionaliteten, der opfylder brugerens anmodninger. Applikationslaget validerer data, anvender forretningsregler og

koordinerer dataflowet mellem præsentationslaget og forretningslaget. Det sikrer, at applikationen fungerer korrekt og leverer de ønskede resultater.

Forretningslaget:

Dette lag udgør kernen i systemets forretningslogik og funktionalitet. Det håndterer og implementerer de processer og regler, der er nødvendige for at opfylde systemets krav og mål. Ved at adskille forretningslogikken fra de øvrige lag skabes der en klar struktur, der gør systemet lettere at vedligeholde, teste og opgradere i fremtiden.

Dataadgangslaget:

Dette lag bruges til at kommunikere med databasen og udfører opgaver som at hente, gemme, opdatere og slette data (CRUD-operationer). Det fungerer som et abstraktionslag mellem applikationen og databasen, hvilket sikrer, at logikken for datahåndtering er centraliseret og adskilt fra de andre lag. Dataadgangslaget håndterer også forbindelsesstyring, optimering af forespørgsler og sikker adgang til data. Denne adskillelse gør det nemmere at vedligeholde koden og muliggør fleksibilitet, hvis datakilden skal ændres eller opgraderes i fremtiden.

Arkitektur og Design

System arkitektur:

Backend

I ASP.NET Core planlægger jeg at anvende en objektorienteret struktur (OOP), som jeg lærte om tidlig under min uddannelse. Jeg synes, med denne tilgang at det giver et bedre overblik over applikationens arkitektur og gør det nemmere at finde og genbruge metoder.

Strukturen vil være opdelt i følgende lag:

Controller:

- Dette lag håndterer API-anmodninger og fungerer som grænseflade mellem frontend og backend. Controllerne koordinerer kommunikationen mellem brugergrænsefladen og de underliggende lag.

Business Layer:

- Her håndteres al forretningslogik, så controllerne forbliver slanke og fokuserede. Dette lag fungerer som et mellemlid, der sikrer, at data behandles korrekt, inden de sendes videre til databasen eller returneres til frontend.

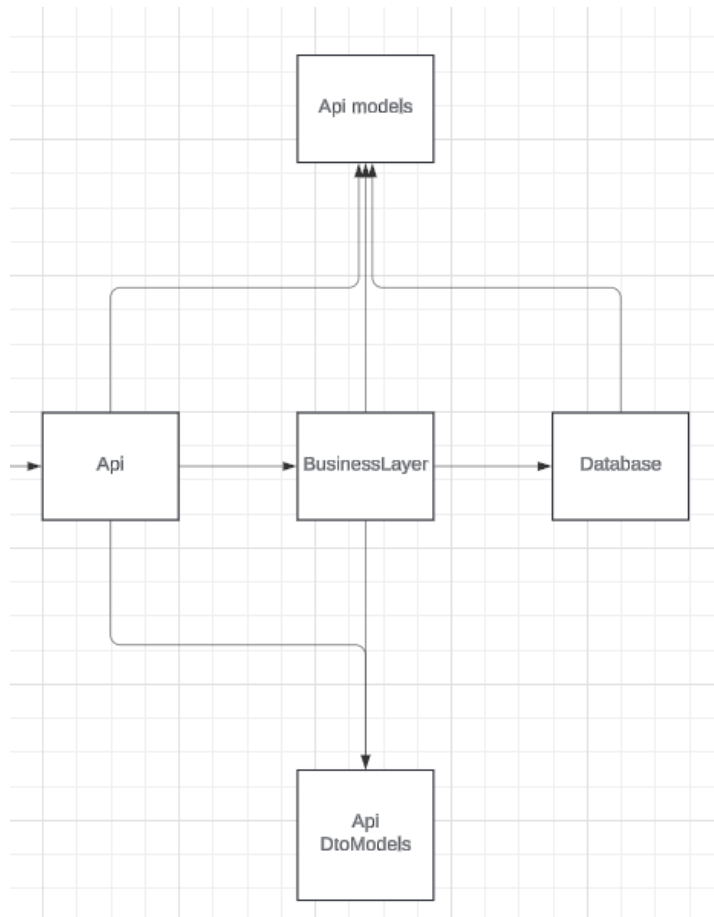
Database Layer:

- Dette lag kommunikerer direkte med databasen og håndterer CRUD-operationer via repository-mønsteret. Det sikrer, at dataadgang er isoleret og genanvendelig.

Modeller og DTO'er:

- **Modeller** repræsenterer databaseentiteter og bruges til at definere skemaet og datarelationerne.
- **DTO'er** bruges til at overføre data mellem lagene og sikrer, at kun relevante data sendes mellem backend og frontend, hvilket reducerer risikoen for unødigt eksponering.

Denne struktur sikrer en klar adskillelse af ansvar og gør applikationen mere vedligeholdelsesvenlig og skalerbar.



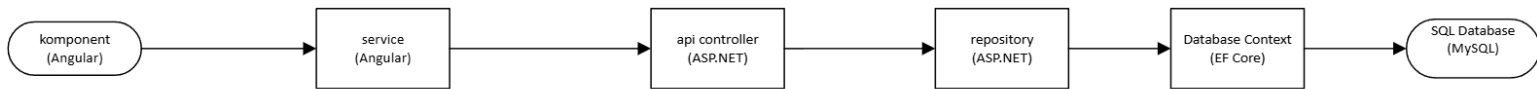
Frontend

I Angular planlægger jeg at anvende en objektorienteret struktur (OOP), hvor følgende roller og ansvarsområder er defineret:

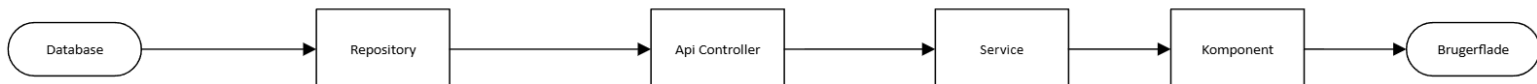
- **Komponenter:** Disse repræsenterer brugergrænsefladen (UI) og indeholder HTML, CSS/SCSS og TypeScript. Komponenter kan betragtes som objekter med deres eget ansvar og tilstand. De kan have klasser, metoder og properties, og deres primære formål er at håndtere præsentationen og interaktionen med brugeren.
- **Services:** Disse fungerer som forretningslaget og håndterer logikken, f.eks. kald til webservices og databehandling, der skal deles mellem flere komponenter. Services gør det muligt at genbruge logik og bliver normalt injiceret i komponenter ved hjælp af dependency injection.

- **Interfaces og klasser:** Disse bruges som modeller til at definere strukturen af data.

api kald

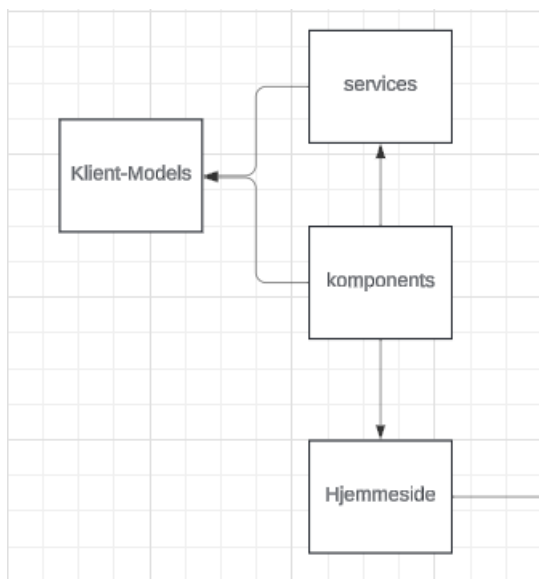


respons



Interfaces sikrer typekontrol, mens klasser kan bruges til at tilføje metoder eller ekstra funktionalitet til datamodellerne.

Denne struktur sikrer en klar adskillelse mellem præsentation, logik og data, hvilket gør koden mere overskuelig, genanvendelig og vedligeholdelsesvenlig.

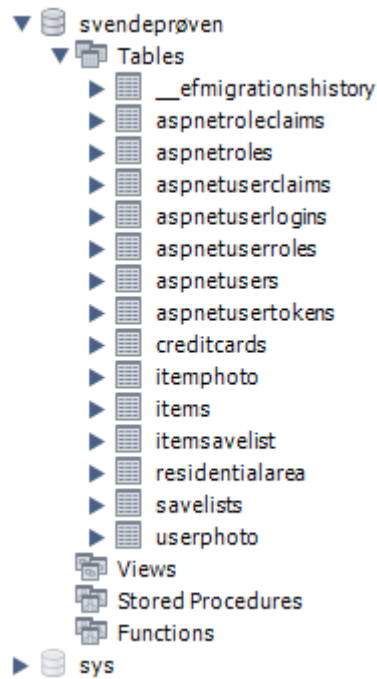


En typisk Api kald og respons vil se ud som følgende:

Database

Til dette projekt har jeg valgt at gøre brug af en cloud server, hvor jeg kan sætte min mysql server op, denne løsning vil kun være godt i en kort periode efter hjemmesiden offentliggjort, efter det skal det en store server til for det.

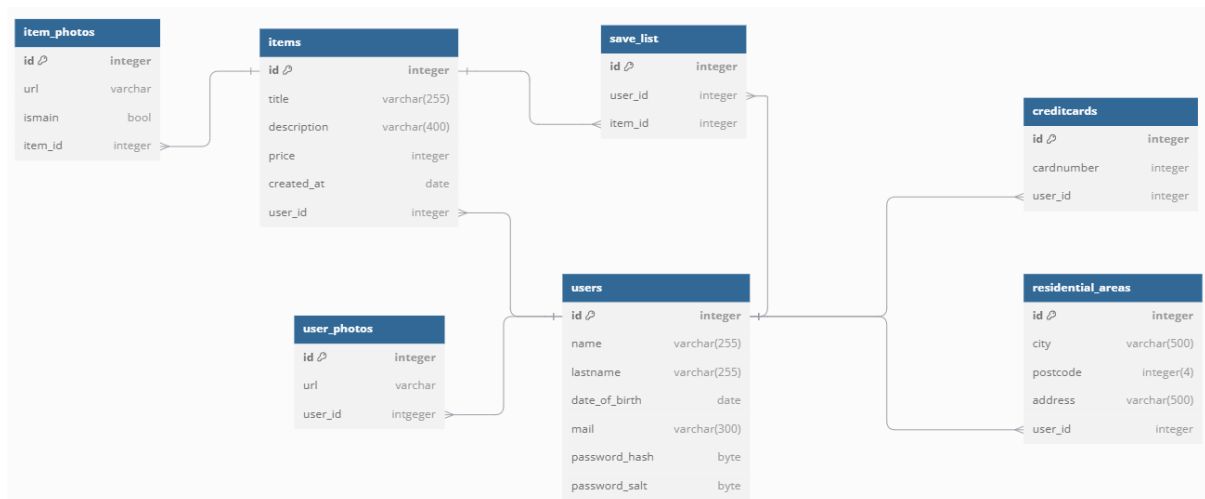
for udviklingsfasen er der installeret MySQL workbench 8.0 ce, men i produktion vil det nok være en god ide at supplementer med andre værktøjer, så som dedikerede overvågnings- og backupværktøjer eller finde noget helt andet der kan erstatte det



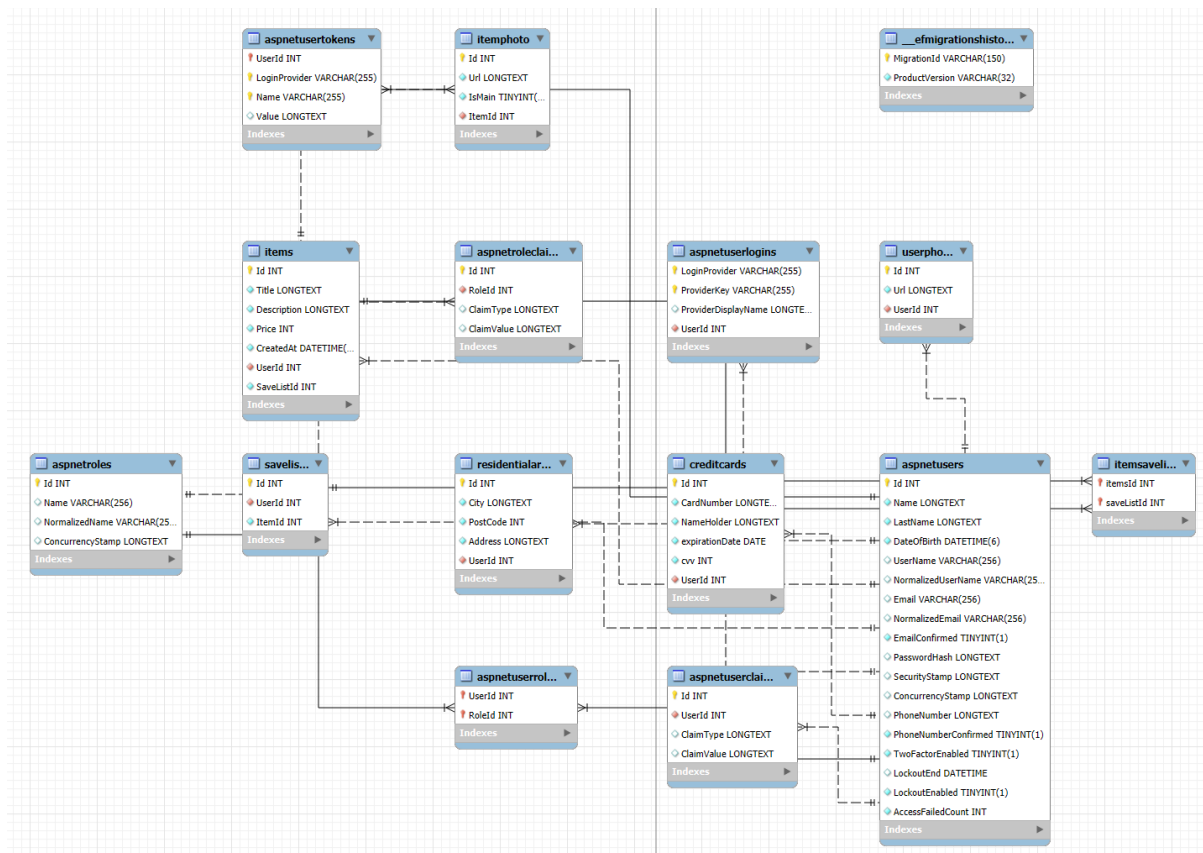
Formål

Databasen formål er at opbevare det nødvendige dataer, så som brugers private oplysninger, deres produkter og gemte annoncer

Her er udarbejdet den originale skabelon inden der blev tilføjet identity til det



Her er et er diagram efter identity ud fra mysql workbench



Relationer

Som man kan se, består den af One-To-One, One-To-Many og Many-To-Many, hvilket nok ikke skal forklares hvorfor, noget som jeg nok kunne havde fra valgt nogle steder på min database skulle nok være nogle af id'erne, det kunne f.eks. være userphoto, creditcard og residentailarea og måske flere, men det er lige fra toppen af hoved, grunden til at der id i alle, er fordi til at starte med vidste jeg ikke bedre og nu hvor jeg er ved slutning vil jeg helst ikke risikere at fjerne det og opleve problemer, jeg skal yderligere fikse.

Implementeringsdetaljer

Backend

Dependency injection

Jeg bruger dependency injection for at have løs købling (Loose Coupling) til at f.eks. let kunne ændre serviceimplementeringen uden at skulle ændre i kontrolleren, det giver også bedre mulighed for testen. Samt at konfigurere dem og administrere dem et sted, i program.cs

Konvertering mellem modeller

jeg bruger dto-modeller og entity-modeller, jeg bruger det til at separere de forskellige lag i mit projekt og til at give det nødvendige data eller fordi, der er noget data der skal sendes, men ikke gemmes i databasen, det kan for eksempel være når vi sender en token ud

```
public class UserDto
{
    2 references
    public string Mail { get; set; }
    2 references
    public string Token { get; set; }
    1 reference
    public int UserId { get; set; }
}
```

det kan man se her at entity-modellen for databasen ikke har token

```
public class User : IdentityUser<int>
{
    5 references
    public string Name { get; set; }
    3 references
    public string LastName { get; set; }
    3 references
    public DateTime DateOfBirth { get; set; }

    0 references
    public List<Item> Items { get; set; }
    1 reference
    public List<CreditCard> Card { get; set; }
    14 references
    public ResidentialArea LivingPlace { get; set; }
    11 references
    public UserPhoto Photo { get; set; }
    1 reference
    public SaveList SaveList { get; set; }

    1 reference
    public ICollection<UserRole> UserRoles { get; set; } = [];
}
```

et eksempel på en dto, hvor der ikke skal sendes alt data med om genstanden kan være

```
public class ShortItemInfoDto
{
    2 references
    public int Id { get; set; }
    2 references
    public string Title { get; set; }
    2 references
    public int Price { get; set; }
    2 references
    public string description { get; set; }
    2 references
    public List<PhotoDto> Photos { get; set; }
}
```

det kan man se hvis vi sammenligner det med entity-modellen for item

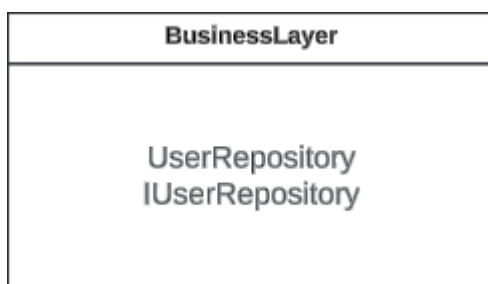
```
public class Item
{
    9 references
    public int Id { get; set; }
    6 references
    public string Title { get; set; }
    6 references
    public string Description { get; set; }
    6 references
    public int Price { get; set; }
    1 reference
    public DateTime CreatedAt { get; set; } = DateTime.UtcNow;

    3 references
    public int UserId { get; set; }
    7 references
    public User user { get; set; }
    0 references
    public int SaveListId { get; set; }
    0 references
    public List<SaveList>? saveList { get; set; }
    12 references
    public List<ItemPhoto> itemPhotos { get; set; }
}
```

vi kan også bruge dto som sikkerhed, da vi kun skal give det data vi skal bruge det kan for eksempel være vores salt til password, det er nok ikke en god ide at sende den til vores applikation, at det kan vores dto-modeller sørge for

Dataadgangslaget: Repositories

I repositories har jeg mine repos og deres interfaces, det vil lægge sådan her for hver repository det laves



DatabaseLayer

For hver * indikere at det er inde i DatabaseLayer

- Database:
Det er databasen der indeholder entity framework core database context
- Migrations:
Det er denne mappe der indeholder alle de migrations som entity framework core har genereret

Models

Jeg bruger models som entity models, der er ikke nogen stor begrundelse til hvorfor, jeg er bare hvad jeg normalt har gjort.

Endpoints

Der er sat swagger up til apien, så man kan se alle de endpoints der er, et eksempel på det kan være /authUser/Login.

Vi giver apien det her data.

```
{
  "mail": "nicolaiaa03@gmail.com",
  "password": "!!Xrq22baw"
}
```

Det så er vores output det her.

```
{
  "mail": "nicolaiaa03@gmail.com",
  "token": "eyJhbGciOiJIUzUxMiIsInR5cCI6Ikp0
    qcQN69jOwoPqcpCY6JXJVKZHf519wHbrnUtJ
  "userId": 2
}
```

Object-relational mapping (ORM)

Jeg har valgt at bruge entity framework core som ORM i projektet, da jeg har kendskab til efCore, der er valgt at lave databasen gennem code-first-metode, da det giver mere kontrol over, hvordan modellerne oprettes

```
public class Database : IdentityDbContext<User, Role, int, IdentityUserClaim<int>, UserRole, IdentityUserLogin<int>, IdentityRoleClaim<int>, IdentityUserToken<int>>
{
    0 references
    public Database(DbContextOptions<Database> options): base(options)
    {
    }

    0 references
    protected override void OnModelCreating(ModelBuilder builder)
    {
        base.OnModelCreating(builder);

        builder.Entity<User>()
            .HasMany(ur => ur.UserRoles)
            .WithOne(u => u.User)
            .HasForeignKey(ur => ur.UserId)
            .IsRequired();

        builder.Entity<Role>()
            .HasMany(ur => ur.UserRoles)
            .WithOne(u => u.Role)
            .HasForeignKey(ur => ur.RoleId)
            .IsRequired();
    }

    4 references
    public DbSet<CreditCard> CreditCards { get; set; }
    4 references
    public DbSet<SaveList> SaveLists { get; set; }
    9 references
    public DbSet<Item> Items { get; set; }
}
```


Frontend | hjemmeside

Indledning

Denne sektion vil blive brugt på at snakke om måder at kode komponenter på, for at få en forståelse på hvorfor komponenter eller andet er lavet på den måde de er

Forms

Forms er en essentiel del af min hjemmeside, da brugerne skal kunne registrere sig, logge ind, samt oprette og ændre indhold. Jeg har valgt at bruge **template-driven forms**, da de passer godt til projektets nuværende krav.

Grunden til mit valg er, at formularerne i projektet ikke er komplekse eller dynamiske, og projektet stadig er relativt lille. Ved at bruge template-driven forms kan jeg håndtere meget af funktionaliteten direkte i HTML-skabelonerne, såsom validering og databinding. Dette gør implementeringen simpel og hurtig, hvilket passer godt til projektets størrelse og behov på nuværende tidspunkt.

```
<form #CreditForm="ngForm" (ngSubmit)="CreateCard()">
  <div class="form-group">
    <div class="input-group">
      <a>kort holders navn</a>
      <input [(ngModel)]="model.nameHolder" name="Name" type="text">
    </div>
    <div class="input-group">
      <a>kort nummer</a>
      <input [(ngModel)]="model.cardNumber" name="CardNumber" type="text" minlength="16" maxlength="16">
    </div>
    <div class="input-group">
      <a>udløbsdato</a>
      <input [(ngModel)]="model.expirationDate" name="ExpirationDate" type="date">
      <a>CVV</a>
      <input [(ngModel)]="model.Cvv" name="Cvv" type="number">
    </div>
  </div>
  <button class="btn btn-primary" >Register</button>
</form>
```

Modals

Jeg bruger modals til at håndtere bekræftelser og ændringer, hvilket gør dem til en vigtig del af brugeroplevelsen. For implementeringen har jeg valgt det eksterne bibliotek **ngx-bootstrap**, da det også bruges til andre komponenter i projektet. Et af de primære argumenter for valget er, at det er nemt at forstå og hurtigt at opsætte efter en kort introduktion i dokumentationen. Denne tilgang gør det muligt at opnå en ensartet og brugervenlig implementering af modals i applikationen.

Her er en modal for bekræftelse

```
<div class="modal-header">
  <h3 class="modal-title pull-left">
    Du er ved at fjerne {{title}} announce fra dine produkter!
  </h3>
  <button
    type="button"
    class="btn-close close pull-right"
    (click)="bsModalRef.hide()"
    <span class="visually-hidden">&times;</span>
  </button>
</div>
<div class="modal-body">
  <h4>er du sikker på du vil fortsætte</h4>
</div>
<div class="modal-footer">
  <button type="button" (click)="bsModalRef.hide()">Annuller</button>
  <button (click)="onDeleteItem()">Slet</button>
</div>
```

Her er en modal for ændringer

```
<div class="modal-header">
  <h4 class="modal-title pull-left">{{item.title}}</h4>
  <button type="button" class="btn-close close pull-right" (click)="bsModalRef.hide()"
    <span class="visually-hidden">&times;</span>
  </button>
</div>
<form action="">
  <div class="modal-body">
    <div class="form-container">
      <div class="input-group">
        <input type="text" [placeholder]="item.title">
        <input type="text" [placeholder]="item.price">
      </div>
      <textarea name="" type="text" [placeholder]="item.description" id=""></textarea>
    </div>
    <hr>
    <h3>tilføj eller fjern billeder</h3>
    <button disabled>add foto</button>
    <!-- foreach loop
    <img src="" alt="">
    -->
  </div>
  <div class="modal-footer">
    <button type="button" (click)="bsModalRef.hide()">Annuller</button>
    <button (click)="update()">Gem</button>
  </div>
</form>
```

Toastr

Jeg bruger et eksternt bibliotek der hedder ngx-toastr, det er hurtigt at sætte med en dependency injektion.

```
private toastr = inject(ToastrService);
```

Jeg bruger det til at give beskeder om fejl, fx hvis du skriver en kort en det gør man let ved at skrive sådan her

```
error: (error) => this.toastr.error(error.error),
```

Eksterne biblioteker

Bootstrap og ngx bootstrap

Jeg bruger dem for at en god mængde komponenter, for at bruge mindre tid på at selv at lave det gennem html og css,

Ngx-toastr

For at hurtig og let at kunne opsætte toastr komponenter

Scss

For at lettere vedligeholde og organisere mine styles.

For at konfigurere dem alle in skal jeg tilføje dem in i styles i angular.json

```
"styles": [  
  "node_modules/bootstrap/dist/css/bootstrap.min.css",  
  "node_modules/ngx-toastr/toastr.css",  
  "src/styles.scss"  
],
```

Strukturering af Webservice og webservice-kald

baseUrl

der er oprettet en string til at gemme en baseUrl, den bruge til at kalde webservices, det vil gøre det let at udskifte den baseUrl man skal bruge ud, hvis det vil blive ændret senere.

Web Service

Et webservice-kald giver et klart overblik over, hvad der foregår, og hvilke data der returneres. Dette gør det nemt at justere, hvad der skal sendes eller modtages, til hvis der kræves mere eller mindre information.

Et eksempel på et **GET-webservice-kald** ser sådan ud:

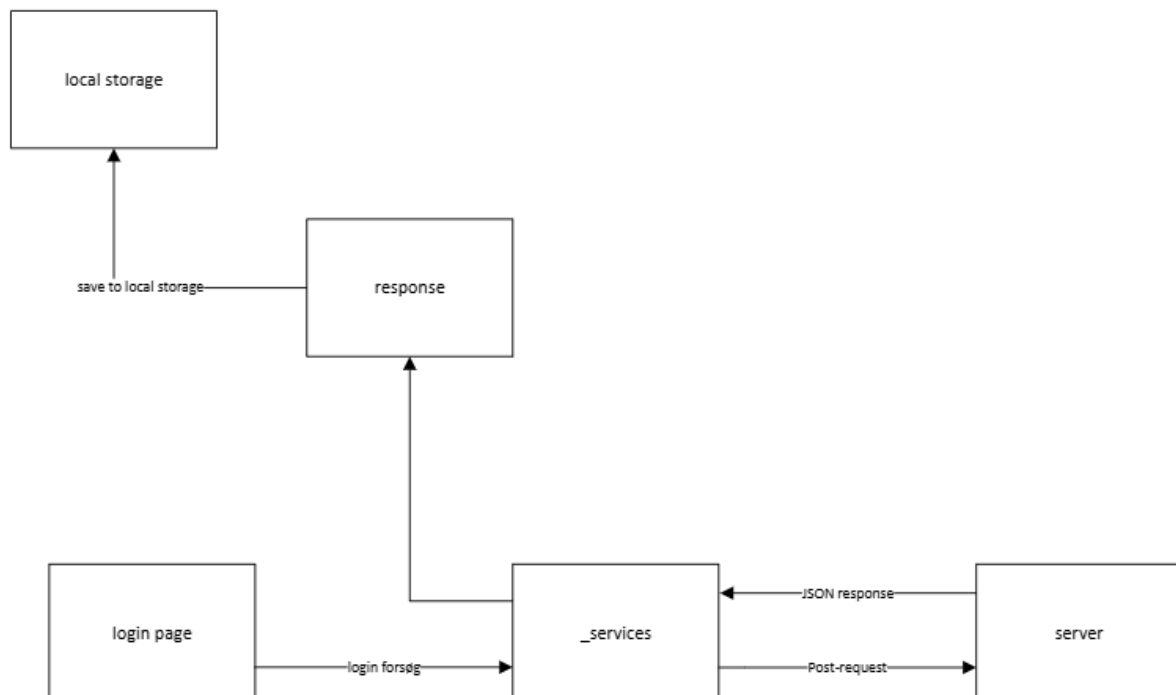
```
getAll(): Observable<item[]> {  
  return this.http.get<item[]>(this.baseUrl + '/Items').pipe(  
    map((items) => {  
      if (items && Array.isArray(items) && items.length > 0) {  
        return items;  
      }  
      return []; //returnere tom array, hvis der map ikke for en item array  
    })  
  );  
}
```

Efter kaldet returneres en **Observable** med pipe. Herefter anvender jeg map til at transformere responsen fra **GET-kaldet**. Hvis der findes mindst én genstand i responsen, returneres disse som items; ellers returneres blot et tomt array.

Authentication

Authentication anvendes i nogle af webkaldende det gøres gennem deres header, hvilket sker ved hjælp af en Bearer-token. For at generere og validere en bruger kræves login og registrering.

Diagrammet viser, hvordan vi opnår en token gennem loginprocessen:



Jeg bruger en http interceptor i angular for at fange og indsætte en Authorization-token ind i anmodninger, så at min api kan validere dem

Måden jeg gjorde det på var at lave en interceptor gennem Angular cli, det gjorde jeg med en har kommando

```
ng generate interceptor jwt
```

Efter det lavede jeg det til en `HttpInterceptor`

Forbinder jeg min `accountservice` op for at kunne få fat i mit signal som sætter min brugers tilstand. Efter jeg har tjekket brugers tilstand, laver jeg en klon af anmodning og sætter authorization headeren med min token og sender min opdaterede anmodning afsted.

```
export const jwtInterceptor: HttpInterceptorFn = (req, next) => {  
  
  const accountService = inject(AccountService);  
  
  if(accountService.CurrentUser()){  
    req = req.clone({  
      setHeaders: {  
        Authorization: `Bearer ${accountService.CurrentUser()?.token}`  
      }  
    })  
  }  
  
  return next(req);  
};
```

for at jeg kan bruge min interceptor skal jeg registrere den i min `app.config`

```
provideHttpClient(withInterceptors([jwtInterceptor])),
```

Kandidater

Frontend

Blazor c#:

Blazor er en Microsoft-teknologi, der bygger webapplikationer ved hjælp af C#.

Fordele:

- **Integreret med Microsoft-økosystem:** hvis du gør brug af andre Microsoft-teknologier, (f.eks. .NET) kan man let komme i gang i Blazor
- **Single Language:** ved brug af c# i frontend og backend kan man forenkle udviklingsprocessen og reducerer behovet for sprogskrift
- **Genbrug af .NET kode og biblioteker:** der er mulighed for at kunne genbruge eksisterende .NET kode og biblioteker, som kan øge produktiviteten og udviklingstiden

Ulemper:

- **Stejl læringskurve:** kan have en svær indlæringskurve for udviklere der lige er begyndt eller der har arbejdet mere med JavaScript, især hvis de ikke har kendskab til .Net eller c#
- **længere starttid og loading:** da Blazor bruger WebAssembly skal de konvertere alt c# kode om til WebAssembly. Da .Net runtime bliver downloadet som en WebAssembly fil og det kan give længere starttid og loading.
- **Mindre gruppe/community:**

Laravel

Laravel er primært en backend-framework men har frontend integration med Blade.

Fordele:

- **Alt i en løsning:** Laravel kan bruges både til backend og frontend med Blade-skabeloner.
- **Simple opsætning:** er god til hvis du vil lave noget simpel og let læringskurve for begyndere.
- **Stort sammenhold/community:** Laravel har mange ressourcer og tutorials.

Ulemper:

- **Ikke optimeret til en tung frontend:** Laravel har mange built-in features, lavet til at lave web udvikling lettere, men hvis du sammenligner det til Angular eller Blazor, har den mindre features og kan være en udfordring at lave en stor og robust hjemmeside.
- **Langsom:** Laravel er ikke det hurtigste framework der kan bruges, så hjemmeside udvikling kan godt tage længere tiden end andre.
- **Opdateres flere gange om måneden:** Laravel for regulært opdateringer, som er godt, men det dårlige ved det er at ældre versioner af produkter kan hurtigt blive buggy.

Angular

Angular er en klient side rendered framework lavet af Google, der har komplekse operationer.

Fordele:

- **Moderne og kraftfuldt:** det kan bruges til komplekse applikationer med masser interaktivitet.
- **Stort fællesskab/community:** der er mange ressourcer, værktøjer og plugins tilgængelig.
- **Typescript:** der gøres brug af type Script som giver bedre udviklingsoplevelse med typekontrol

- **God struktur:** der følges en form for oop struktur i Angular, som kan være velegnet til store projekter med mange features

Ulemper:

- **Svær læringskurve:** det kan være svært at lære Angular som en nu begynder.
- **Problemer med at opdatere til nyere versioner:** det kan godt være en udfordring at skulle ændre mellem store opdateringer, da det kan f.eks. give problemer med eksterne biblioteker, samt opdatering af kode.
- **Kompleks:** da Angular godt kan være kompleks, er det bedst til store projekter med komplekse systemer.

Konklusion:

Jeg har valgt at bruge Angular, da det tilbyder kraftfulde værktøjer og eksterne biblioteker, som passer godt til mine behov. Jo mere tid jeg bruger på Angular, jo mere værdsætter jeg dets fleksibilitet og muligheder. Selvom jeg også kunne have valgt Blazor, føler jeg, at Angular giver mig mere kontrol og frihed i udviklingsprocessen. Desuden oplever jeg, at de udfordringer, jeg støder på i Angular, som regel kan løses, hvilket gør det til en pålidelig løsning for mit projekt.

Backend

Asp.net core c#

Asp.net er en microsoft-teknologi, der gør brug af c# og har mange eksterne biblioteker/nuget pakker, som man kan bruge til at gøre udvikling hurtigere

Fordele:

- **Modent økosystem:** asp.net core er en moden teknologi med en omfattende dokumentation
- **Out-Of-The-Box:** asp.net kan have hurtig udvikling med funktioner som scaffolding, indbyggede authentication værktøjer og integration med Entity Framework Core.
- **God ydeevne:** koden er optimeret til moderne applikationer og kan håndtere højt trafikniveau
- **Sikkerhed:** indbyggede mekanismer til sikkerhed som Identity Framework og autentificering – og autoriseringsintegrationer.

Ulemper:

- **Avanceret konfiguration:** det kan være svært at konfigurere custom middleware, authentication eller andre features, hvis man ikke har så meget erfaring om det
- **Brug mere hukommelse:** .NET core applikationer kan godt bruge mere hukommelse end andre alternativer.
- **Mindre hjælp til små projekter:** asp.net core har stort fokus på enterprise niveauet, så det kan godt føles svært at finde noget for mindre eller forsøgsprojekter

Rust

Fordele:

- **Ydeevne:** Rust er hurtig og bliver sammenligneligt med c/c++ på grund af dens lave niveau af abstraktion og kompilering til native kode.

- **Sikkerhed:** Rust eliminerer typiske fejl som nul pointer dereferences og data races ved compile time og har en god fokus på hukommelses sikkerhed uden brug af garbage collection.
- **Lav ressourceforbrug:** er god til applikationer, der skal være ressourceeffektivitet og have en god ydeevne.
- **Cross-platform:** kan køre på næsten enhver platform med minimal konfiguration.

Ulemper:

- **Svært?:** rust er kendt for at være svært at lære, især for nybegyndere, f.eks. "borrow checker" kan udfordrer, hvis man ikke er vant til memory management.
- **Lille fællesskab:** rust har en mindre udviklerbase og fællesskab, hvilket ville kunne gøre det svære for at finde ressourcer og tutorials.
- **Udviklingshastighed:** på grund af den strenge compile-time kontrol kan udvikling tage længere tid sammenlignet med andre sprog.

Konklusion:

Jeg vælger Asp.net core det jeg har brugt det i flere tidligere projekter og har en god forståelse på de nuget pakker, jeg har tænkt mig at skulle bruge og de features der allerede eksisterer i asp.net core, også selvom jeg synes det lyder interessant med rust, ved jeg helst ikke bruge for lang tid på apien.

Server

MS Sql

Fordele:

- **God integration med Microsoft-økosystemet:** fungerer godt med Asp.net, Azure og andre Microsoft-værktøjer
- **Høj ydeevne:** optimeret til komplekse forespørgsler og store datamængder.
- **Brugervenlighed:** intuitive værktøjer som SQL Server Management Studio (SSMS).

Ulemper:

- **Ressourcekrævende:** Kræver mere hardware og ressourcer sammenlignet med letvægtsalternativer.
- **Hosting:** MSSQL, kan let have problemer, hvis man kører det på andet en Windows maskine.

My SQL

Fordele:

- **Gratis og open source:** tilgængelig uden licensomkostninger.
- **Fleksibilitet;** kan bruges på tværs af flere platforme.
- **Letvægt:** lavere ressourceforbrug, hvilket gør det velegnet til mindre projekter
- **Stort fællesskab/community:** mange open source-værktøjer der understøtter MySQL

Ulempe:

- **Mindst sikkerhed:** for at få det bedste sikkerhed på mysql skal man selv konfigurere de sikkerhedsaspekter, samt manuel opsætte dem
- **Afhængighed af tredjepartsmoduler:** for nogle funktioner og optimeringer er man ofte nødt til at bruge tredjepartsværktøjer, hvilket kan øge kompleksiteten.
- **Fragmentering:** over tid kan MySQL-tabeller blive fragmenterede, hvilket kan påvirke ydeevnen og kræver regelmæssig vedligeholdelse

Konklusion:

Jeg vil bruge MySQL, den største årsag er at jeg bruger en server, der kører med ubuntu og har selv oplevet at MS SQL ikke kunne køre på det, selvom har jeg også brugt det for og man kan også få nuget pakker til at bruge Entity Framework core til MySQL.

Diagrammer

Da min Klasse diagrammer er for store har jeg valgt at give links til det i stedet

Api:

https://lucid.app/lucidchart/41efb447-b412-4cec-8901-175e0183cebc/edit?view_items=G2Qq-OrShk~p&invitationId=inv_8efb8549-392f-4439-abeb-876fea854b4f

Front end:

https://lucid.app/lucidchart/b1400108-82bf-4520-9999-66c27f69bd7b/edit?viewport_loc=-3446%2C-790%2C4490%2C2178%2CHWEp-vi-RSFO&invitationId=inv_986f3947-7f86-4e80-a6a4-a6aee0f22a70

Flowcharts

[flowcharts.vsdX](#)

Obs. Hvis nogle af dem ikke dur vil de også lægge i bilag

Server

Cloudserveren ville være tilgængelig 24/7, til og med den 15. december 2024

Brugernavn og adgangskode for at adgang til My Sql databasen gennem f.eks. My Sql Workbench eller gennem andre

Brugernavn:

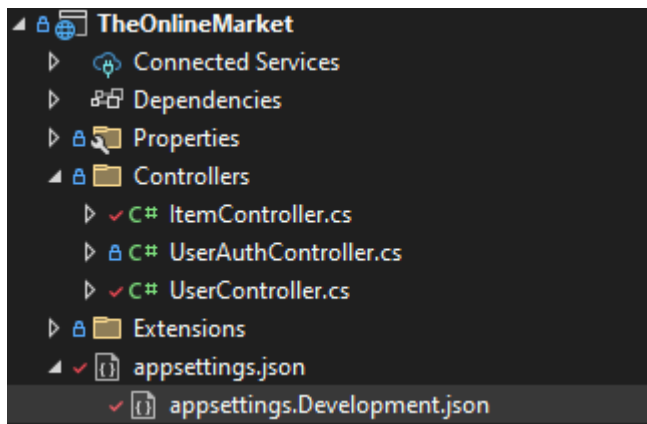
Syzygy

Password:

!!Xrq22baw

Forbindelses streng

Forbindelses strengen skal sættes i appsettings.Development.json den kan man finde gennem TheOnlineMarket->apssetings.json->appsettings.development.json



inde i den vil der så en "defaultConnection": du erstatter "forbindelses streng her" med

```
{
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft.AspNetCore": "Warning"
    }
  },
  "ConnectionStrings": {
    "DefaultConnection": "forbindelses streng her" //strengen befinder sig i produktrapporten under Linket
  }
}
```

Linket

"server=185.249.225.22;Port=3306;Database=Svendeprøven;User=Syzygy;Password=!!Xrq22baw;"

Så vil den se sådan her ud

```
{
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft.AspNetCore": "Warning"
    }
  },
  "ConnectionStrings": {
    "DefaultConnection": "server=185.249.225.22;Port=3306;Database=Svendeprøven;User=Syzygy;Password=!!Xrq22baw;"
  }
}
```

Så snart det er gjort kan du, kan du starte apien

links

Github repositories

Frontend

<https://github.com/Syzygy622f/OnlineMarket>

Backend

<https://github.com/Syzygy622f/TheOnlineMarket>

Trello

<https://trello.com/invite/b/6744909938a187b90bdb78a8/ATTI0490e4b72d901cbaafb33a5df246c1fd2F5D7F16/onlinemarketplace>

Design

<https://www.figma.com/design/HIZ1kToMZey6EaV9CxnW64/Online-Market-Place?t=j2DOWiVKDVYknxPh-1>